Studiengang:	Informatik
Prüfer:	Prof. Dr. Hans-Joachim Bungartz
Betreuer:	Dr. rer. nat. Stefan Zimmer
CR-Klassifikation:	F.2.1, G.1.2, H.3.3

Diplomarbeit Nr. 2264

Data Mining mit Dünnen Gittern

Dirk Pflüger

begonnen am:	01. Oktober 2004
beendet am:	01. April 2005

Fakultät für Informatik, Elektrotechnik und Informationstechnik
Institut für Parallele und Verteilte Systeme
Abteilung Simulation großer Systeme
Universität Stuttgart
70569 Stuttgart

Inhaltsverzeichnis

1 Einführung

We are drowning in information, but starved for knowledge.
John Naisbitt

Die vergangenen Jahrzehnte verzeichnen ein immer rasanteres Wachstum in den Fähigkeiten, Daten zu generieren, zu sammeln und zu speichern. Fortschritte in den verschiedensten Bereichen von Wissenschaft und Technik machen immer größere Datenbestände verfügbar.

Satelliten und Weltraumteleskope senden stündlich große Mengen an Daten zur Erde. Weltweit beobachten bereits weit über 100.000 Sensoren an Wetterstationen, Handelsschiffen, Flugzeugen oder Bojen auf und unter der Meeresoberfläche unsere Erde. Doch nicht nur in globalen Maßstäben fallen große Datenmengen an. Als Beispiel sei die Medizin erwähnt: Hier entstehen in der Forschung, wie beim Humangenomprojekt, oder bei Untersuchungsergebnissen, wie der Computertomografie, große Datenbestände.

Aus wirtschaftlicher Sicht gefragter sind Daten, die durch die zunehmende Elektronisierung ganzer Lebensbereiche und durch die weltweite Vernetzung gesammelt werden können: Das Kundenverhalten beim Online-Shopping wird aufgezeichnet, um Direktmarketing zu betreiben. Anwender versuchen, sich mit Spam-Filtern gegen unerwünschte Werbesendungen zu wehren. Banken bemühen sich, die Kreditwürdigkeit von Kunden bereits im Vorfeld abzuschätzen. Hinzu kommt die günstige Verfügbarkeit immer größerer Speichermedien. Das hohe kommerzielle Interesse beflügelt den Handel mit Daten jeglicher Art.

Auch wenn viele Daten zur Verfügung stehen, so ist oft wenig darüber bekannt. Auf herkömmliche Art und Weise kann nur Information aus Datenbanken abgerufen werden, indem eine direkten Anfrage formuliert wird. Verborgene und unbekannte Information ist auf diesem Wege nicht oder nur schwer auffindbar.

Die Methoden und Verfahren des Knowledge Discovery bzw. des Data Mining versuchen, diese Lücken zu füllen und unbekannte Information zu erschließen und nutzbar zu machen.

Diese Arbeit fokusiert auf das Teilgebiet der maschinellen Klassifikation. Hierzu wird ein Ansatz mittels sogenannter dünner Gitter vorgestellt, der auf der Diskretisierung des Merkmalraumes basiert. Er löst die selbe Formulierung des Klassifikationproblems wie einige der populärsten Klassifikationsalgorithmen.

In Kapitel 2 wird dieser Ansatz in den Kontext eingeordnet und gegenüber verwandten Methoden abgegrenzt. Kapitel 3 und 4 erläutern die benötigten Grundlagen und führen die Technik der dünnen Gitter ein. In Kapitel 5 wird die Realisierung einer Testumgebung zur Klassifikation mittels dünner Gitter vorgestellt. Eine effiziente Implementierung wird diskutiert und die Funktionsweise des entstandenen Paketes erläutert. Kapitel 6 zeigt die Verwendbarkeit des Systems anhand einiger Testbeispiele. Abschließend werden in Kapitel 7 die wichtigsten Ergebnisse zusammengefasst und ein kurzer Ausblick gegeben.

2 Data Mining & Knowledge Discovery

In diesem Kapitel werden die benötigten Grundbegriffe wie *Data Mining* oder *maschinelles Lernen* erläutert und gegeneinander abgegrenzt. Die Realisierung der Klassifikation von Daten mittels Regression wird näher betrachtet und als Minimierungsproblem in der Formulierung der Regularisierungstheorie vorgestellt. Gemeinsamkeiten mit ähnlichen Verfahren werden dargestellt und eine geeignete Diskretisierung wird eingeführt, die zur Verwendung dünner Gitter führt.

2.1 Überblick

Die Unterschiede zwischen den Termini *Data Mining* und *Knowledge Discovery* sind häufig unklar und verschwommen. Oft werden beide Begriffe synonym als Oberbegriffe verwendet für alles, was mit der Extraktion von Wissen in Zusammenhang gebracht werden kann. Der Begriff *Knowledge Discovery in Databases* (kurz *KDD*) wird eher von Seiten der Statistik, der Datenanalyse und der Managementinformationssysteme verwendet, *Data Mining* hingegen findet hauptsächlich in den Kreisen der *Künstlichen Intelligenz* und des *Maschinellen Lernens* (kurz *ML*) Verwendung.

Nach [FPSSU96] wurde der Begriff KDD gewählt, um den Gesamtprozess des Findens und Extrahierens von Information und Wissen aus Datenmengen zu beschreiben, während Data Mining für die Anwendung von Algorithmen zum Gewinnen von Modellen, Mustern oder Strukturen aus Daten steht. Data Mining ist nach dieser Definition einer der Schritte auf dem Weg des KDD von der Datenwüste zum konkreten Wissen.

Knowledge discovery in databases wurde definiert als „der nicht-triviale Prozess des Identifizierens von gültigen, neuen, potenziell nützlichen und letztlich verständlichen Mustern in Daten“ [FPSSU96], bzw. „die nicht-triviale Extraktion von impliziter, vormals unbekannter und potentiell nützlicher Information aus Daten“ [FrPSM92]. Der KDD-Prozess beinhaltet sämtliche Schritte, von der Identifikation der Problemstellung, über eine Vorauswahl der Daten, die Vorverarbeitung (Bereinigung der Daten, Filtern, Rauschunterdrückung, Reduktion der Dimensionalität, ...), das Data Mining und die Interpretation der erkannten Muster, Strukturen und Abhängigkeiten, bis hin zur Verwertung und Umsetzung der Ergebnisse (siehe auch Abbildung 2.1).

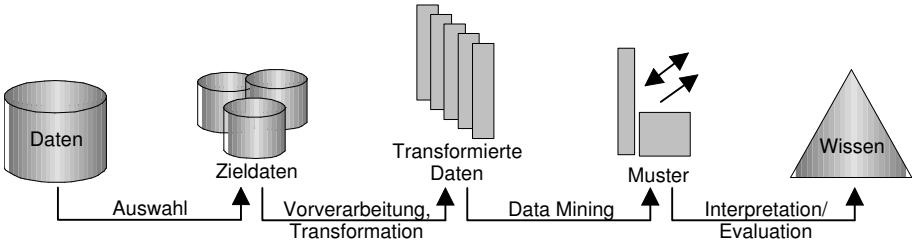


Abbildung 2.1: Skizze der wichtigsten Schritte des *Knowledge Discovery*-Prozesses.

Selbstverständlich ist dieser Ablauf keine Einbahnstraße; das Wissen aus jedem Schritt kann verwendet werden, um zu einem vorherigen Schritt zurückzukehren und diesen zu verfeinern oder abzuändern. Wird beispielsweise erkannt, dass die Auswahl der Attribute zu restriktiv war, so sollte diese erneut und unter Berücksichtigung dieser Erkenntnis getroffen werden. Im Folgenden werden die Schritte vor und nach dem Data Mining nicht weiter betrachtet.

2.2 Data Mining

Data Mining kann in mehrerere Aufgabengebiete unterteilt werden. [HaMS01] unterscheidet in *explorative Datenanalyse*, *Finden von Mustern und Regeln*, *inhaltsbasierte Suche*, sowie in *deskriptives* und *prediktives Modellieren*. Auch hier sind die Grenzen zwischen den verschiedenen Begriffen fließend.

Der Schwerpunkt der explorativen Datenanalyse ist es, interaktives und visuelles Durchsuchen („Browsen“) von (hochdimensionalen) Datenbeständen zu ermöglichen. Ziel des Findens von Mustern und Regeln ist beispielsweise das Auffinden von Anomalitäten, wie die Erkennung von ungewöhlichen Sternkonstellationen in astronomischen Aufnahmen oder das Aufstellen eines Satzes von Regeln, der für den menschlichen Benutzer verständlich ist, wie im *Advanced Scout*-System [BCPP+97], das Statistiken von NBA-Spielen für Trainer verständlich auswertet. Bei der inhaltsbasierten Suche sind ähnliche Datensätze von Interesse. Sie beschränkt sich jedoch nicht auf die reine textuelle Suche, wie sie von Internetanwendungen bekannt ist. Denkbar sind ebenso die Suche nach Bildern anhand einer natürlichsprachen Beschreibung oder ähnliche Anwendungsgebiete.

Beim deskriptiven Modellieren werden Strukturen und Muster in vorhandenen Daten beschrieben – oft bezogen auf alle Daten. Im Allgemeinen wird es benutzt, um aussagekräftige Teilgruppen zu identifizieren, bzw. um die Daten in k Teilgruppen zu unterteilen. *Clustering*-Verfahren sind typische Beispiele hierfür. Das prediktive Modellieren verwendet eine Teilmenge der Attribute, um explizite Werte für andere Attribute vorherzusagen. *Klassifikations-* und *Regressions*-Verfahren werden meist zu diesem Aufgabengebiet gezählt, obwohl sie zum Teil ebenfalls verwendet werden können, um Daten in Gruppen oder Cluster zu unterteilen.

Oft wird nur in die zwei Hauptziele des *deskriptiven* und *prediktiven Modellierens* unterschieden, und das Finden von Mustern und Regeln wird zum deskriptiven Modellieren gezählt. Verfahren, die sich hauptsächlich mit der interaktiven grafischen Darstellung oder mit reinen Suchverfahren beschäftigen werden außen vor gelassen.

2.3 Klassifikation und Regression

Klassifikationsverfahren lernen eine Funktion, die ein Datum einer von mehreren vorgegebenen Klassen zuordnet. Gegeben sei eine Menge vorklassifizierter Daten, die Trainingsdaten

$$S = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times K\}_{i=1}^{M},$$

wobei K die Menge der möglichen Klassen ist. Zu jedem dieser M Datenpunkte $\mathbf{x}_i$ aus dem d-dimensionalen Merkmalsraum existiert eine Klassifikation, ein Klassenlabel y_i. Das Ziel ist neuen, unklassifizierten Daten, den sogenannten Testdaten, jeweils die richtige Klasse zuzuordnen.

Ein typisches Beispiel für Klassifikationsverfahren sind Entscheidungsbäume, die anhand von Fallunterscheidungen in jeweils einem Attribut die Klassenzugehörigkeit bestimmen.

Die Blätter des Baumes entsprechen den Klassen, Verzweigungen im Baum Verknüpfungen zwischen einzelnen Attributen. Abbildung 2.2 zeigt einen einfachen Entscheidungsbaum, in dem einem Satz Wetterdaten die Fälle „Tennis spielen“ und „nicht Tennis spielen“ zugeordnet werden. Ein realistischeres Szenario ist das einer Bank, die automatisch anhand von Kundendaten die Kreditwürdigkeit eines Kundens entscheiden möchte. Die maschinelle Erkennung handgeschriebener Ziffern ist ein weiteres Beispiel.

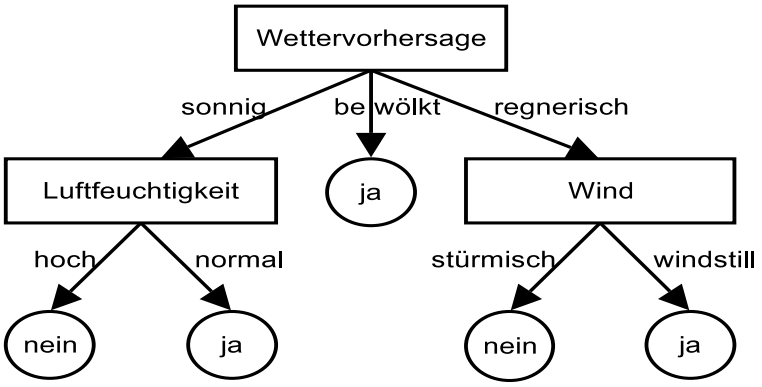


Abbildung 2.2: Entscheidungsfindung, ob Tennis gespielt werden soll, anhand von Wetterdaten.

Weitere typische Vertreter von maschinellen Lernverfahren zur Klassifikation sind neben Entscheidungsbäumen beispielsweise *k-nächste Nachbarn*, *Neuronale Netze* oder *Support-Vektor-Maschinen*.

Im Beispiel der Bank, die zukünftige Kunden automatisch in „kreditwürdig“ (k) und „nicht kreditwürdig“ (nk) unterscheiden möchte, wären die Trainingsdaten beispielsweise Kundendaten von früheren Kunden. Ein dreidimensionaler Merkmalsraum könnte aus der Höhe des monatlichen Bruttogehalts, der Dauer der letzten Anstellung und der Höhe der Verschuldung zur Zeit des Kreditantrages bestehen. Ordnet man jedem dieser Datenpunkte die in der Vergangenheit gefällte Entscheidung zu, ob ein Kredit vergeben wurde oder nicht, so ist $K = \{\mathrm{k}, \mathrm{nk}\}$, also $y_i \in \{\mathrm{k}, \mathrm{nk}\}$.

Eine verwandte Sichtweise liegt den Regressionsverfahren zugrunde: Sie lernen eine Funktion, die zu einem Datum einen reellen Wert berechnet. Ausgangspunkt ist das Wissen oder die Annahme, dass zwischen den Merkmalen $x_1, \ldots, x_d$ und y ein funktionaler Zusammenhang besteht. Gegeben sei eine Menge von Merkmalen und der zugehörige Funktionswert:

$$S = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^{M}.$$

Es wird angenommen, dass die Daten zufälligen Funktionsauswertungen einer unbekannten Funktion f entstammen, die durch Rauschen etwas verzerrt wurden. Gesucht wird die unbekannte Funktion $y = f(x_1, x_2, \ldots, x_d)$, die zu einem Funktionenraum V gehört, der über $\mathbb{R}^d$ definiert wurde.

Da Klassifikationsverfahren ebenfalls als Funktionsapproximation aufgefasst werden können ist der Hauptunterschied zu Regressionsverfahren, dass der Funktionswert y bei der Klassifikation diskret und bei der Regression kontinuierlich ist. Es ist leicht einsichtig, dass aus Regressionsverfahren Verfahren zur Klassifikation werden, indem der reelle Ausgabebereich in geeignete Teilintervalle unterteilt wird und diese auf vorgegebene Klassen abgebildet werden. Klassifikation kann aus dem Blickwinkel der Regression als Approximation von verstreuten Daten betrachtet werden.

Viele Algorithmen fallen daher unter beide Kategorien. Während beispielsweise Entscheidungsbäume und k-nächste Nachbarn meist nur eine Klassenzuordnung zurückgeben kön-

nen, arbeiten Neuronale Netze und Support-Vektor-Maschinen als Klassifikatoren mit reellen Werten und einer Abbildung dieser auf die Klassenlabel.

2.4 Regularisierung

Die Klassifikation mit dünnen Gittern folgt dem Regressionsansatz. Gelernt wird ein Zweiklassen-Problem: Jedes Datum wird genau einer von zwei Klassen zugeordnet, meist den beiden Werten -1 und 1. Die gesuchte Funktion $f : \mathbb{R}^d \to \mathbb{R}$, der Klassifikator, soll die Trainingsdaten

$$S = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}\}_{i=1}^{M}$$

in einem Funktionenraum V, der über $\mathbb{R}^d$ definiert ist, approximieren. Als zusätzliche Anforderung sollte f jedoch auch für die noch ungesehenen Testdaten gute Ergebnisse liefern. Stehen für die beiden Klassen die Werte $+1$ und -1, so kann in positive und negative Werte unterschieden werden: Alle Daten mit Funktionswert ≥ 0 werden der Klasse $+1$, alle mit Funktionswert < 0 der Klasse -1 zugeordnet.

Die Regressionsaufgabe ist für die praktisch interessanten Fälle schlecht gestellt: Normalerweise ist $\dim(V)$ viel größer als M, so dass es unendlich viele Interpolanten gibt. Nach [Hada23] muss für ein gut gestelltes Problem eine Lösung existieren, diese eindeutig sein und stetig von den Daten abhängen. Um eine der Lösungen auszuwählen, muss *a priori* Wissen über die Funktion angenommen werden. Der gebräuchlichste Ansatz ist die Annahme einer gewissen Glattheit der Funktion f.

Im Hinblick auf die Testdaten wird für den Klassifikator ohnehin ein Kompromiss gesucht zwischen der Genauigkeit der Approximation, also der Größe des Approximationsfehlers, und der gewünschten Generalisierungsfähigkeit bezüglich der Testdaten. Wichtig ist, dass die Klassenzugehörigkeit stimmt, d. h. dass bei Verwendung der Klassenlabel $+1$ und -1 der Nulldurchgang an der richtigen Stelle ist; der exakte Funktionswert ist von geringerem Interesse. Insbesondere soll das Rauschen in den Trainingsdaten nicht zu sogenanntem *Overfitting* führen. Overfitting beschreibt das Problem der Überanpassung eines Modells an einen Datensatz: Beschreibt die gelernte Funktion die Trainingsdaten inklusive aller Störungen exakt, so oszilliert sie eventuell zu stark und kann nicht mehr zur Prognose verwendet werden. Es liegt daher nahe, eine gewisse Glattheit der gesuchten Funktion anzunehmen.

2.4.1 Ansatz der Regularisierungstheorie

Der Ansatz der *Regularisierungstheorie* ist, dass das schlecht gestellte Problem des Lernens von Beispielen durch eine Variationsformulierung gelöst werden kann, die sowohl die Trainingsdaten als auch Vorwissen über die Glattheit der gesuchten Funktion enthält [TiAr77]. Hierfür wird ein zusätzlicher Regularisierungsterm eingeführt. Gesucht wird die Lösung von

$$\min_{f \in V} H[f],$$

das Minimum $f \in V$ von

$$H[f] = \frac{1}{M} \sum_{i=1}^{M} \mathcal{V}(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_K^2. \tag{2.1}$$

Dabei ist $\mathcal{V}$ eine Fehlerfunktion, die den Approximationsfehler misst, d. h. die Abweichung von der tatsächlichen zur gewünschten Klassifikation. Der Regularisierungssoperator $\|f\|_K^2$,

auch Stabilisator genannt, ist ein Glattheitsfunktional. Minimiert wird nicht mehr allein der Fehler, sondern ein Trade-Off zwischen Kosten und Glattheit, gewichtet mit dem Regularisierungsparameter λ. Die Wahl des genauen Wertes für λ kann über *Kreuzauswertung* (cross-validation) erfolgen [Alle74, GoHW79] oder über Techniken wie *strukturelle Risiko-Minimierung* [Vapn82].

2.4.2 Verwandte Verfahren

Die allgemeine Formulierung 2.1 ist von besonderem Interesse, da sich verschiedene Neuronale Netze sowie Support-Vektor-Maschinen (SVM) als Regularisierungsansatz formulieren lassen [EvPP00]: Sie entsprechen der Minimierung von H mit jeweils unterschiedlicher Wahl der Kostenfunktion $\mathcal{V}$:

- Klassische (L_2) Regularisierungsnetzwerke (RN) sind Neuronale Netze mit einer verdeckten Schicht an Neuronen. Die Kostenfunktion hierfür ist

$$\mathcal{V}(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2.$$

- Support-Vektor-Maschinen zur Regression verwenden die Kostenfunktion

$$\mathcal{V}(y_i, f(\mathbf{x}_i)) = |y_i - f(\mathbf{x}_i)|_\epsilon,$$

  wobei $|\cdot|_\epsilon$ Vapniks epsilon-insensitive Norm ist, mit

$$|x|_\epsilon = \begin{cases} 0 & \text{falls } |x| < \epsilon, \\ |x| - \epsilon & \text{sonst,} \end{cases}$$

  d. h. Fehler kleiner als ϵ werden nicht gezählt. Aus Sicht dieser Kostenfunktion ist eine Funktion f, die näher als ϵ an allen Daten liegt, ein optimaler Interpolant.

- Für Support-Vektor-Maschinen zur Klassifikation ist

$$\mathcal{V}(y_i, f(\mathbf{x}_i)) = |1 - y_i f(\mathbf{x}_i)|_+,$$

  mit

$$|x|_+ = \begin{cases} x & \text{falls } |x| \geq 0, \\ 0 & \text{sonst.} \end{cases}$$

  Im Unterschied zu SVMs zur Regression, die einen beliebigen reellen Wert für die y_i der Trainingsdaten zulassen, ist für die SVMs zur Klassifikation der Wert von y_i je nach Klasse $+1$ oder -1. Interessanterweise wurde die Technik der Support-Vektor-Maschinen zuerst zur binären Klassifikation entwickelt [Vapn95]. Erst später wurde sie auf die allgemeinere Aufgabenstellung der Regression erweitert.

Kostenfunktion und Stabilisator müssen so gewählt werden, dass die Lösung existiert und eindeutig ist.

- Je nach Kernel-Funktion K ergeben sich verschiedene Typen von Regularisierungsnetzwerken, beispielsweise Gaußsche *Radial Basis Function Networks* (RBFN) für

$$K(\mathbf{x} - \mathbf{y}) = e^{(-\|\mathbf{x}-\mathbf{y}\|^2)}$$

  oder ein Multilayer Perzeptron mit

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\mathbf{x} \cdot \mathbf{y} - \theta)$$

  (nur für einige Werte von θ).

- Auch für SVMs lassen sich verschiedene Kernel-Funktionen verwenden. Beispiele sind Gaußscher Kernel
$$K(\mathbf{x}, \mathbf{y}) = e^{(-g \cdot \|\mathbf{x}-\mathbf{y}\|^2)},$$
Skalarprodukt-Kernel
$$K(\mathbf{x}, \mathbf{y}) = s \cdot (\mathbf{x}, \mathbf{y}),$$
polynomieller Kernel für Polynomgrad p
$$K(\mathbf{x}, \mathbf{y}) = (s \cdot \mathbf{xy} + r)^p$$
oder Sigmoid-Kernel
$$K(\mathbf{x}, \mathbf{y}) = \tanh(s \cdot \mathbf{xy} + r).$$

- Allgemein formuliert ist $\|f\|_K^2$ eine Norm in einem *Reproducing Kernel Hilbert Space* (RKHS) V [Aron50]. Die Kernel-Funktion K muss positiv definit sein, definiert den RKHS und induziert die Norm $\|f\|_K^2$.

Zur Klassifikation müssen Regressionsfunktionen mit der Schwellwertfunktion $\text{sign}(f(\mathbf{x}))$ auf die beiden Klassen abgebildet werden. Aus statistischer Sicht wird mit dem quadratischen Fehler den Datenpunkten mit der höchsten Wahrscheinlichkeit das größte Gewicht gegeben und nicht den Punkten, die für die Unterscheidung von Daten in die beiden Klassen am wichtigsten sind, also die, die am Rand zwischen zwei Gebieten mit unterschiedlicher Klasse liegen. Die Kostenfunktionen der SVMs hingegen sprechen Punkten am Rand eine höhere Bedeutung zu.

2.5 Diskretisierung des Raumes

Die meisten Lernverfahren zur Klassifikation arbeiten datenzentriert; beispielsweise werden bei RBFNs radiale Basisfunktionen auf den Trainingsdaten zentriert. Der Ansatz für die Klassifikation mit Dünnen Gittern diskretisiert den Raum und wird im Folgenden näher erläutert. Die Notation orientiert sich an [GaGT01].

Der Raum $\mathcal{H}$ wird eingeschränkt auf einen endlich dimensionalen Teilraum V_N, der von der Basis $\Phi_N = \{\phi_i\}_{i=1}^N$ aufgespannt wird. Die gesuchte Funktion $f_N \in V_N$ setzt sich somit aus einer gewichteten Summe der Basisfunktionen ϕ_i mit den Freiheitsgraden $\{\alpha_i\}_{i=1}^N$ zusammen:
$$f_N(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j \phi_j(\mathbf{x}). \tag{2.2}$$

Verfolgt wird der klassische Ansatz der Regularisierungsnetzwerke mit der Kostenfunktion
$$\mathcal{V}(y_i, f(\mathbf{x}_i)) := (y_i - f(\mathbf{x}_i))^2. \tag{2.3}$$

Für den Regularisierungsoperator gelte im Weiteren
$$\|f\|_K^2 := \|\nabla f\|_{L_2}^2. \tag{2.4}$$

Gleichungen 2.3 und 2.4 eingesetzt in 2.1 führt zu dem Problem der Minimierung von
$$H[f_N] = \frac{1}{M} \sum_{i=1}^{M} (y_i - f_N(\mathbf{x}_i))^2 + \lambda \|\nabla f_N\|_{L_2}^2 \tag{2.5}$$

mit $f_N \in V_N$. Die Funktion f_N (2.2) eingesetzt ergibt

$$H[f_N] = \frac{1}{M}\sum_{i=1}^{M}\left(y_i - \sum_{j=1}^{N}\alpha_j\phi_j(\mathbf{x}_i)\right)^2 + \lambda\left\|\nabla\sum_{j=1}^{N}\alpha_j\phi_j\right\|_{L_2}^2 \tag{2.6}$$

$$= \frac{1}{M}\sum_{i=1}^{M}\left(\sum_{j=1}^{N}\alpha_j\phi_j(\mathbf{x}_i) - y_i\right)^2 + \lambda\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j(\nabla\phi_i, \nabla\phi_j)_{L_2} \tag{2.7}$$

Gesucht wird das Minimum von H in Abhängigkeit von den Freiheitsgraden α_i. Ableiten nach α_k, $k = 1, \ldots, N$ liefert N Gleichungen der Form

$$0 = \frac{\delta H[f_N]}{\delta\alpha_k} \tag{2.8}$$

$$= \frac{2}{M}\sum_{i=1}^{M}\left(\sum_{j=1}^{N}\alpha_j\phi_j(\mathbf{x}_i) - y_i\right)\phi_k(\mathbf{x}_i) + 2\lambda\sum_{j=1}^{N}\alpha_j(\nabla\phi_j, \nabla\phi_k)_{L_2} \tag{2.9}$$

$$= \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N}\alpha_j\phi_j(\mathbf{x}_i)\phi_k(\mathbf{x}_i) - \frac{1}{M}\sum_{i=1}^{M}y_i\phi_k(\mathbf{x}_i) + \lambda\sum_{j=1}^{N}\alpha_j(\nabla\phi_j, \nabla\phi_k)_{L_2} \tag{2.10}$$

$$= \lambda\sum_{j=1}^{N}\alpha_j(\nabla\phi_j, \nabla\phi_k)_{L_2} + \frac{1}{M}\sum_{j=1}^{N}\alpha_j\sum_{i=1}^{M}\phi_j(\mathbf{x}_i)\phi_k(\mathbf{x}_i) - \frac{1}{M}\sum_{i=1}^{M}y_i\phi_k(\mathbf{x}_i) \tag{2.11}$$

und somit, für $k = 1, \ldots, N$,

$$\sum_{j=1}^{N}\left(M\lambda\alpha_j(\nabla\phi_j, \nabla\phi_k)_{L_2} + \sum_{i=1}^{M}\phi_j(\mathbf{x}_i)\phi_k(\mathbf{x}_i)\right)\alpha_j = \sum_{i=1}^{M}\phi_k(\mathbf{x}_i)y_i. \tag{2.12}$$

Dieses Lineare Gleichungssystem mit N Gleichungen und N Unbekannten α_j, $j = 1, \ldots, N$ lässt sich in Matrixschreibweise darstellen als

$$\left(\lambda C + B \cdot B^T\right)\boldsymbol{\alpha} = By \tag{2.13}$$

mit den Matrizen C, $c_{j,k} = M(\nabla\phi_j, \nabla\phi_k)_{L_2}$, $j, k = 1, \ldots, N$ und B, $b_{j,i} = \phi_j(\mathbf{x}_i)$, $j = 1, \ldots, N$ und $i = 1, \ldots, M$. Die Matrix C verkörpert das Glattheitsfunktional, während die Matrix B (zusammen mit dem Ergebnisvektor y) für die Messung des Fehlers zuständig ist.

C ist ein diskretisierter Laplace-Operator mit homogenen Neumann-Randbedingungen und wird positiv semidefinit sein. B^T ist die Auswertung in den Trainingspunkten $\mathbf{x}_1, \ldots, \mathbf{x}_M$. $\lambda C + B \cdot B^T$ wird durch die Wahl von V_N positiv definit sein und das LGS somit eine eindeutige Lösung besitzen.

2.6 Das Problem der Dimensionalität

Das diskrete Gleichungssystem (2.13) könnte mit Hilfe einer herkömmlichen Finite Elemente Diskretisierung aufgestellt und durch ein geeignetes iteratives Verfahren gelöst werden. Hierfür würden sich Konjugierte Gradienten oder Multigrid Methoden anbieten. Eine Diskretisierung mittels eines regulären uniformen Gitters stößt jedoch bereits bei wenigen Dimensionen auf den sogenannten Fluch der Dimensionalität: Bei n Basisfunktionen bzw.

Gitterpunkten in einer Dimension sind es bei d Dimensionen n^d Basisfunktionen. Mit optimalen Lösungsverfahren ist der Aufwand $\mathcal{O}(n^d)$; die exponentielle Abhängigkeit von d macht es unmöglich, auch nur moderat dimensionierte Probleme zu rechnen. Eine deutliche Verbesserung lässt sich durch die Diskretisierung mit sogenannten Dünnen Gittern erzielen.

In den folgenden beiden Kapiteln wird der verwendete Ansatzraum erläutert. Es wird eine geeignete Basis vorgestellt, die diesen aufspannt. Sie basiert auf einer hierarchischen Struktur der Basisfunktionen und ermöglicht die Technik der dünnen Gitter, die näher betrachtet wird.

3 Ansatzraum und Basis

Als Ansatzraum werden Räume von d-linearen Ansatzfunktionen $\mathbb{R}^d \to \mathbb{R}$ betrachtet, vorerst in einer Dimension. Um diese aufzuspannen werden geeignete Basisfunktionen eingeführt. Dazu werden die Begriffe *hierarchische Basis* und *hierarchischer Überschuss* erläutert. Des weiteren wird das Randwertproblem diskutiert und eine geeignete Modifikation der Basis vorgestellt.

3.1 Der Ansatzraum

Als Vorstufe der Klassifikation werden die Daten bei Lernverfahren im Allgemeinen auf $\Omega = [0,1]^d$ normalisiert. Dies ist bei einer endlichen Menge an Trainings- und Testdaten durch eine geeignete Skalierung des Merkmalraums durchführbar. Der Klassifikator $f : [0,1]^d \to \mathbb{R}$ soll dann die Trainingsdaten $S = \{(\mathbf{x}_i, y_i) \in [0,1]^d \times \{-1,1\}\}_{i=1}^M$ in einem Funktionenraum V, der über $[0,1]^d$ definiert ist, approximieren.

Sei, im Hinblick auf die spätere Definition der hierarchischen Basis, $N := 2^l$ für ein Level $l \in \mathbb{R}$ und sei $V_N \subset V$ der Ansatzraum der stückweise linearen Funktionen mit Dimension $d = 1$ und Maschenweite $h_l := 1/N = 2^{-l}$. Zur Vereinfachung seien die Randwerte null, d. h. $f_N(0) = f_N(1) = 0, \forall f_N \in V_N$.

Eine Standardbasis, die V_N aufspannt, ist die Knotenbasis. Grundlage für die Basisfunktionen ist die Standard-Hutfunktion auf dem Intervall $[-1,1]$,

$$\phi(x) := \begin{cases} 1 - |x| & \text{falls } x \in [-1,1], \\ 0 & \text{sonst.} \end{cases}$$

Durch Translation und Stauchung lassen sich die Basisfunktionen in Abhängigkeit von dem Level l und einem Index i ausdrücken als

$$\phi_{l,i}(x) := \phi\left(\frac{x - i \cdot h_l}{h_l}\right) = \phi\left(\frac{x - i \cdot 2^{-l}}{2^{-l}}\right) = \phi\left(x \cdot 2^l - i\right). \tag{3.1}$$

Auf Level $l = 3$ ergibt dies für die Maschenweite $h = \frac{1}{2^l} = \frac{1}{8}$ sieben Basisfunktionen $\phi_{3,i}(x)$, $i = 1, \ldots, N-1$, mit jeweils einem Träger der Breite $2h$, siehe Abbildung 3.1.

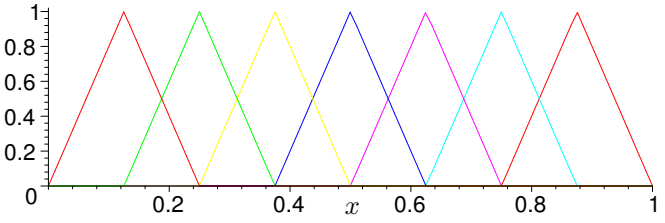


Abbildung 3.1: Knotenbasis für $l = 3$.

An den Knickstellen der stückweise linearen Funktion f_N ist immer nur eine der Basisfunktionen ungleich null. Zur Darstellung der gewichteten Summe ist es daher nur nötig, die

Gewichte α_i an den Abszissen $x_i = i \cdot h$ heranzuziehen. Abbildung 3.2 zeigt das bekannte Bild für die Interpolation einer Parabel mit der Knotenbasis. Die gestrichelte Linie ist der Interpolant $f_N = \sum_{i=1}^{N-1} \alpha_i \phi_{3,i}(x)$.

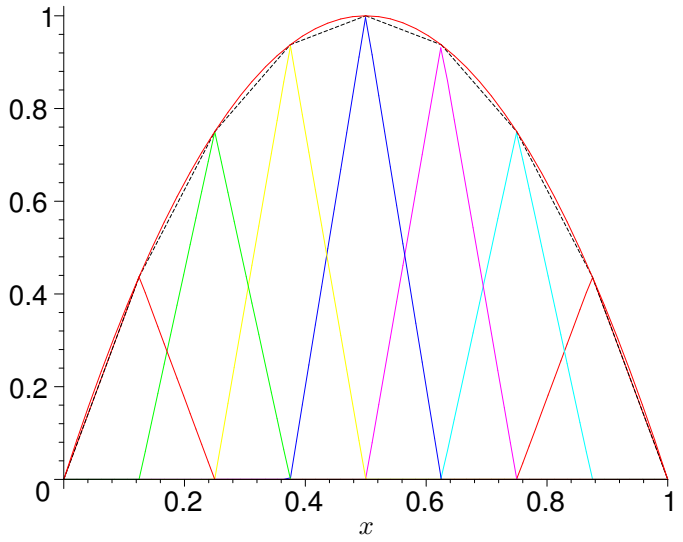


Abbildung 3.2: Interpolation der Parabel $f(x) = 4x(1-x)$ mit der Knotenbasis für $l = 3$.

3.2 Hierarchische Basis

Die Idee einer *hierarchischen Basis* geht bereits auf Archimedes zurück. Er approximierte die Fläche unter einer Parabel, indem er Dreiecke hineinlegte, deren Flächen sich einfach berechnen lassen.

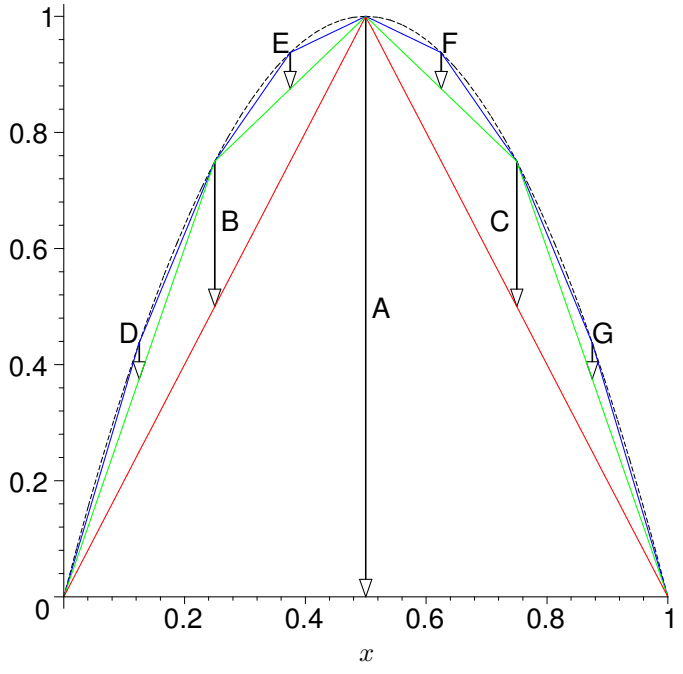


Abbildung 3.3: Quadratur nach Archimedes.

Abbildung 3.3 zeigt die Annäherung der Fläche unter der Parabel durch Dreiecksflächen. Zuerst wird Dreieck A unter die Parabel gelegt und als erste Näherung für die Fläche der Parabel betrachtet. In einer ersten Verfeinerung werden auf Dreieck A die beiden Dreiecke B und C aufgesetzt. Die Näherung für die Parabelfläche entspricht jetzt der Fläche unter der stückweise linearen Funktion f_4 mit Maschenweite $1/4$, die die Parabel an den Stützstellen $1/4$, $1/2$ und $3/4$ interpoliert, beziehungsweise der Summe der Flächen der drei Dreiecke. In einer weiteren Verfeinerung entstehen vier weitere Dreiecke, D bis G. Die obersten Kanten der aufeinander gesetzten Dreiecke ergeben den selben Interpolanten f_8 wie die Knotenbasis in Abbildung 3.2.

Dieses Prinzip der Verfeinerung lässt sich rekursiv fortsetzen. Mit Rekursionstiefe (oder Level) l erhält man $2^l - 1$ Dreiecke, Dreieck A liegt auf Level 1. Beim Schritt von Level $l-1$ zu Level l entstehen 2^{l-1} neue Dreiecke. Wird jedes dieser Dreiecke auf die x-Achse geschert, so lassen sich die Dreiecke durch die bereits eingeführten Hutfunktionen (Gleichung 3.1) darstellen, siehe Abbildung 3.4.

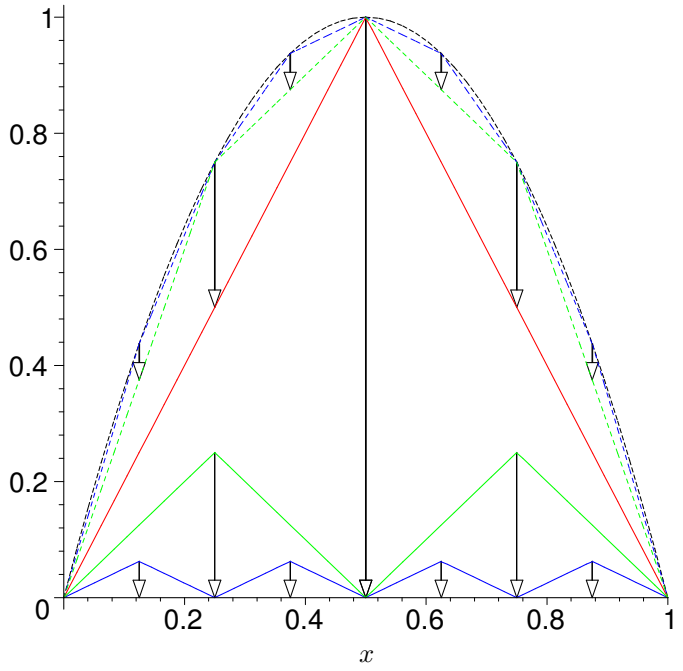


Abbildung 3.4: Scherung der Dreiecke auf die x-Achse.

Dreieck A entspricht der einzigen Funktion auf Level 1, $\phi_{1,1}(x)$, und sei mit Faktor $\alpha_{1,1}$ gewichtet. Die beiden Dreiecke B und C entsprechen allen Hutfunktionen mit ungeraden Indizes auf Level 2, $\alpha_{2,1} \cdot \phi_{2,1}(x)$ und $\alpha_{2,3} \cdot \phi_{2,3}(x)$, die Dreiecke D bis G analog auf Level 3 den Funktionen $\alpha_{3,1} \cdot \phi_{3,1}(x)$, $\alpha_{3,3} \cdot \phi_{3,3}(x)$, $\alpha_{3,5} \cdot \phi_{3,5}(x)$ und $\alpha_{3,7} \cdot \phi_{3,7}(x)$. Die Koeffizienten der Basisfunktionen $\alpha_{l,i}$ werden wie die zugehörigen Basisfunktionen mit Level l und Index i indiziert.

Die Menge der Hutfunktionen

$$\Phi_l := \left\{ \phi_{l',i} : l' \leq l, i \leq 2^{l'} - 1 \text{ und } i \text{ ungerade} \right\}$$

bildet die *hierarchische Basis* bis Level l, die den Raum der stückweise linearen Funktionen mit Maschenweite $h = 1/2^l$ aufspannt. Abbildung 3.5 zeigt die hierarchische Basis für $l = 3$. Φ_l und die Knotenbasis auf Level l haben die selbe Anzahl an Basisfunktionen und definieren den gleichen Funktionsraum.

Ein wesentlicher Unterschied zwischen beiden ist die Berechnung des Funktionswertes des Interpolanten an den Stützstellen oder Gitterpunkten. In Tabelle 3.2 ist dies dargestellt

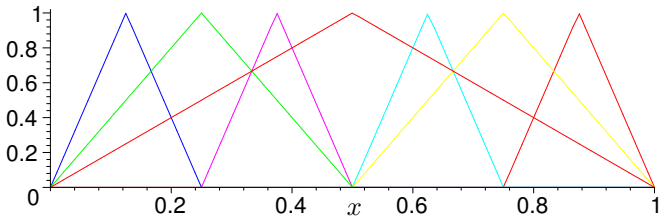


Abbildung 3.5: Hierarchische Basis für $l = 3$.

für die Parabel $f(x) = 4x(1-x)$ mit Diskretisierung auf Level 3 für die Knotenbasis und die hierarchische Basis.

Gitterpunkt	$\frac{1}{8}$	$\frac{2}{8}$	$\frac{3}{8}$	$\frac{4}{8}$	$\frac{5}{8}$	$\frac{6}{8}$	$\frac{7}{8}$
Knotenbasis	$\alpha_{3,1}$	$\alpha_{3,2}$	$\alpha_{3,3}$	$\alpha_{3,4}$	$\alpha_{3,5}$	$\alpha_{3,6}$	$\alpha_{3,7}$
hier. Basis	$1/4 \cdot \alpha_{1,1}$	$2/4 \cdot \alpha_{1,1}$	$3/4 \cdot \alpha_{1,1}$	$\alpha_{1,1}$	$3/4 \cdot \alpha_{1,1}$	$2/4 \cdot \alpha_{1,1}$	$1/4 \cdot \alpha_{1,1}$
	$1/2 \cdot \alpha_{2,1}$	$\alpha_{2,1}$	$1/2 \cdot \alpha_{2,1}$		$1/2 \cdot \alpha_{2,3}$	$\alpha_{2,3}$	$1/2 \cdot \alpha_{2,3}$
	$\alpha_{3,1}$		$\alpha_{3,3}$		$\alpha_{3,5}$		$\alpha_{3,7}$

Tabelle 3.1: Funktionswerte für Level 3 für Knotenbasis und hierarchische Basis an den Gitterpunkten/Interpolationsstellen.

Für die Knotenbasis ist an jedem Gitterpunkt nur eine Basisfunktion ungleich null. Da sich die gesuchte Funktion, der Interpolant, aus der Summe der gewichteten Basisfunktionen ergibt, genügt es, den Koeffizienten dieser heranzuziehen: Die zugehörige Basisfunktion hat dort ihren Spitzenwert 1.

Bei der hierarchischen Basis funktioniert das Ablesen nur am Gitterpunkt 1/2 für die größte Basisfunktion. An allen anderen Gitterpunkten setzt sich der Funktionswert zusammen aus dem Koeffizienten der entsprechenden Basisfunktion und den gewichteten Koeffizienten aller Basisfunktionen niedrigeren Levels, deren Träger die betrachtete Stelle enthalten. Dies erfolgt ganz analog zum Addieren der Dreiecksflächen bei der Quadratur nach Archimedes. Der Wert eines Koeffizienten wird *hierarchischer Überschuss* genannt, da er sich aus der Differenz von tatsächlichem Funktionswert und dem Funktionswert des Interpolanten auf dem vorherigen Level ergibt. In den Abbildungen 3.3 und 3.4 werden die hierarchischen Überschüsse durch Pfeile dargestellt.

Ein bedeutender Vorteil der hierarchischen Basis ist die im Konstruktionsschema enthaltene Adaptivität. Bei der Annäherung an eine hinreichend glatte Funktion werden die Koeffizienten schnell kleiner. Wird die Basis rekursiv erstellt kann der aktuelle Koeffizient verwendet werden um abzuschätzen, ob weiter verfeinert werden soll oder abgebrochen werden kann. Die hierarchische Struktur der Basis kann als Binärbaum aufgefasst werden. Tabelle 3.2 zeigt die Koeffizienten am Beispiel der Interpolation der Parabel $f(x) = 4x(1-x)$. Die Größe der Koeffizienten weist hier sehr schön auf die Größenordnung des Fehlers bei der Interpolation mit stückweise linearen Funktionen hin, $\mathcal{O}(h^2)$.

$\alpha_{1,i}$							1								
$\alpha_{2,i}$				$\frac{1}{4}$							$\frac{1}{4}$				
$\alpha_{3,i}$		$\frac{1}{16}$				$\frac{1}{16}$				$\frac{1}{16}$				$\frac{1}{16}$	
$\alpha_{4,i}$	$\frac{1}{64}$		$\frac{1}{64}$		$\frac{1}{64}$		$\frac{1}{64}$		$\frac{1}{64}$		$\frac{1}{64}$		$\frac{1}{64}$		$\frac{1}{64}$

Tabelle 3.2: Koeffizienten der Basisfunktionen der hierarchischen Basis mit Level 4 zur Interpolation von $f(x) = 4x(1-x)$.

Für die Berechnung der Funktionswerte an den Gitterpunkten genügt eine Traversierung des Binärbaumes, bei der die hierarchischen Überschüsse aufaddiert werden.

3.3 Randbetrachtung

Bisher wurde die Interpolation von Funktionen mit Randwert null betrachtet. Bei der Interpolation von beliebigen Funktionen f auf dem Intervall $[0,1]$ mit den bisher vorgestellten Basen ist der Fehler am Rand $f(0)$ bzw. $f(1)$. Werden die Daten, wie bei der Klassifikation üblich, auf $[0,1]^d$ normalisiert, so liegt mindestens ein Datum auf jeder Intervallgrenze. Möchte man diese nicht immer der Klasse $+1$ zuordnen, so müssen die Randwerte berücksichtigt werden.

Bei Verwendung der Knotenbasis bietet es sich an, die beiden Hutfunktionen $\phi_{l,0}(x)$ und $\phi_{l,2^l}(x)$, die zu den Gitterpunkten auf dem Rand gehören, hinzuzunehmen. Abbildung 3.6 veranschaulicht dies für Level $l = 2$ für die Interpolation eines Polynoms zweiten Grades.

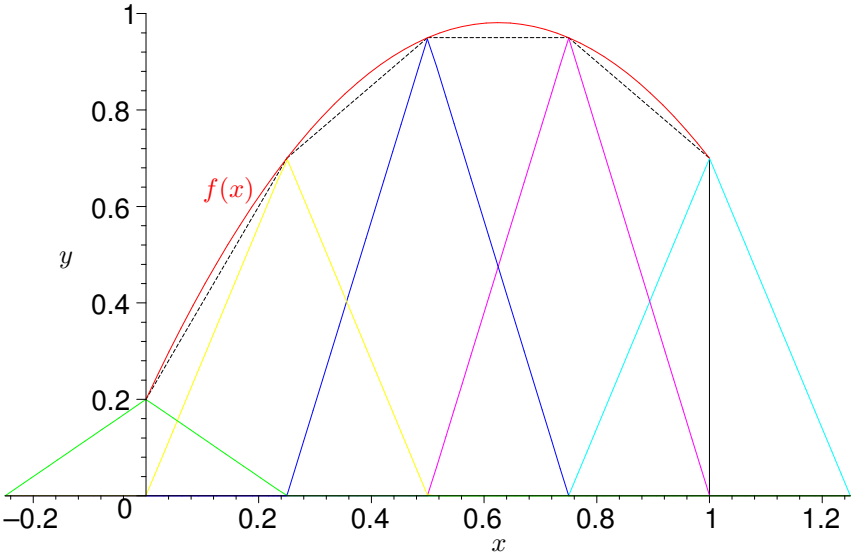


Abbildung 3.6: Interpolation mit der Knotenbasis für Randwerte ungleich null.

Eine Möglichkeit der Berücksichtigung der Randwerte bei Verwendung der hierarchischem Basis ist, unter die Funktion f ein Trapez zu legen, das auf der x-Achse aufliegt und die oberen beiden Eckpunkte $(0, f(0))$ und $(1, f(1))$ besitzt. In Abbildung 3.7 wird es dargestellt durch die gestrichelte rote Linie. Auf dieses Trapez wird die hierarchische Basis aufgesetzt. Algorithmisch gesehen ändert sich für die Berechnung der Funktionswerte nur wenig: Es müssen lediglich die beiden äußeren Randwerte bei der Traversierung des Baumes der Basisfunktionen berücksichtigt werden.

Statt des Trapezes lassen sich auch die beiden Funktionen

$$\phi_{0,0}(x) := \begin{cases} 1-x & \text{falls } x \in [0,1] \\ 0 & \text{sonst} \end{cases} \quad \text{und} \quad \phi_{0,1}(x) := \begin{cases} x & \text{falls } x \in [0,1] \\ 0 & \text{sonst} \end{cases}$$

verwenden. Sie entsprechen zwei Hutfunktionen mit einem Träger der Breite 2, die auf die Randpunkte zentriert und auf das Intervall $[0,1]$ eingeschränkt sind, und ergeben zusammen das Trapez.

Eine zweite Möglichkeit besteht in der Modifikation der Basisfunktionen. Auf jedem Level gibt es *Randbasisfunktionen*. Dies sind die Basisfunktionen $\phi_{l,1}$ und $\phi_{l,2^l-1}$, die am nächs-

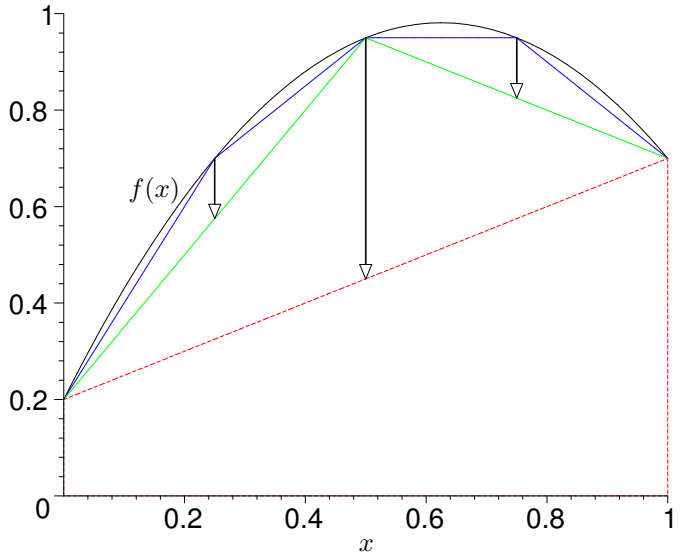


Abbildung 3.7: Interpolation mit der hierarchischen Basis für Randwerte ungleich null, Trapezansatz.

ten bei 0 bzw. 1 liegen. Sie können verwendet werden, um zu den Gebietsgrenzen hin zu extrapolieren. Auf Level 1 ist $\phi_{1,1}$ Randfunktion zu beiden Rändern. Es bietet sich an, für $\phi_{1,1}$ die konstante Funktion $f(x) = 1$ zu wählen. Gewichtet mit $\alpha_{1,1}$ kann die erste Basisfunktion sämtliche konstante Funktionen darstellen.

Auf jedem höheren Level können die beiden Basisfunktionen $\phi_{l,1}$ und $\phi_{l,2^l-1}$ zum Rand hin „ausgeklappt“ werden. Aus den Hutfunktionen werden dann lineare Funktionen mit dem selben Träger, die den Randwert 2 und wie bisher den Wert 1 am Gitterpunkt besitzen. Gleichung 3.2 stellt die modifizierten Basisfunktionen in Abhängigkeit von Level l und Index i dar, Abbildung 3.8 liefert eine anschauliche Darstellung für die ersten vier Level.

$$\phi_{l,i}(x) := \begin{cases} 1 & \text{falls } l = 1 \text{ und } i = 1 \\ \left\{ \begin{array}{ll} 2 - 2^l \cdot x & \text{falls } x \in \left[0, \frac{1}{2^{l-1}}\right] \\ 0 & \text{sonst} \end{array} \right\} & \text{falls } l > 1 \text{ und } i = 1 \\ \left\{ \begin{array}{ll} 2^l \cdot x + 1 - i & \text{falls } x \in \left[1 - \frac{1}{2^{l-1}}, 1\right] \\ 0 & \text{sonst} \end{array} \right\} & \text{falls } l > 1 \text{ und } i = 2^l - 1 \\ \phi\left(x \cdot 2^l - j\right) & \text{sonst} \end{cases} \tag{3.2}$$

Die Basis Φ_l – die Menge aller modifizierten Basisfunktionen der hierarchischen Basis bis zum Level l – spannt nun nicht mehr den Raum der stückweise linearen Funktionen mit Maschenweite $h = 2^{-l}$ auf. Dies gilt nur noch innerhalb des Intervalls $[2h, 1-2h]$. Im Gegenzug können dafür mit gleich vielen Freiheitsgraden oder Gitterpunkten die beiden Randpunkte interpoliert werden. Soll der selbe Ansatzraum wie bei dem Trapezansatz erhalten werden, so müssen zur Basis Φ_l die beiden Randbasisfunktionen des Levels $l+1$ hinzugenommen werden, vergleiche Abbildung 3.9. Dann allerdings entfällt der Vorteil der beiden eingesparten Gitterpunkte. Dieser Gedanke wird in Kapitel 4.3 wieder aufgegriffen werden.

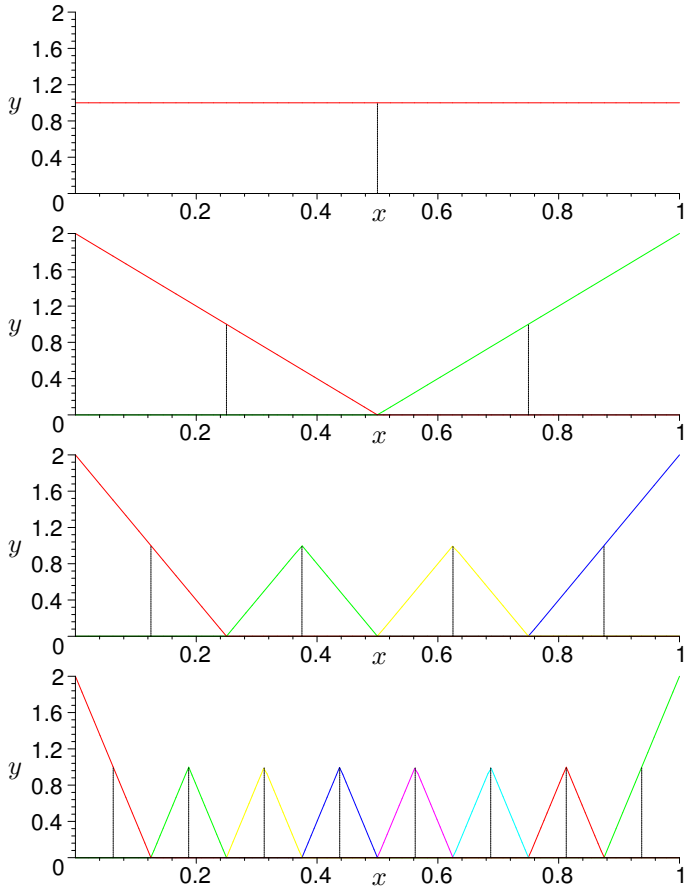


Abbildung 3.8: Hierarch. Basis mit ausgeklappten Basisfunktionen, Level 1 bis 4.

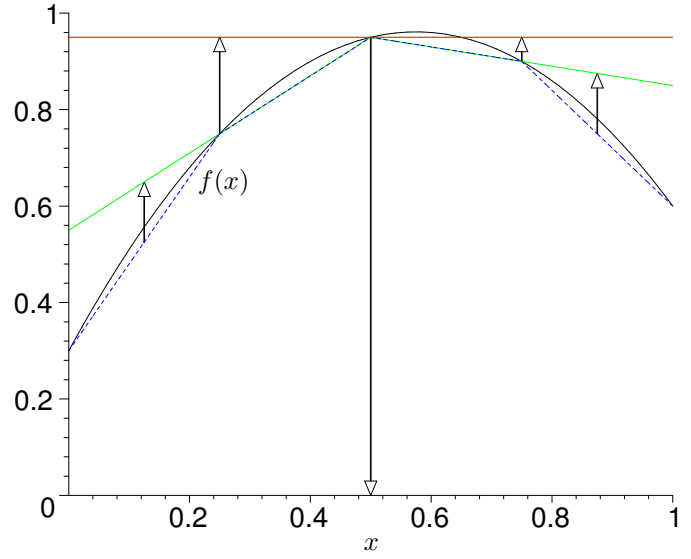


Abbildung 3.9: Interpolation mit der hierarchischen Basis für Randwerte ungleich null, ausgeklappte Basisfunktionen.

4 Höhere Dimensionalität

Der Ansatzraum der stückweise linearen Funktionen im Eindimensionalen soll nun erweitert werden auf den Raum der stückweise d-linearen Funktionen in d Dimensionen. Die Konstruktion der höherdimensionalen Basisfunktionen aus den vorgestellten eindimensionalen Versionen wird vorgestellt und die benötigten Notationen eingeführt.

4.1 Stückweise d-linearer Funktionsraum in d Dimensionen

Es werde weiterhin der normalisierte Merkmalsraum $\Omega = [0,1]^d$ betrachtet. Seien $\mathbf{l}$ und $\mathbf{i}$ Multi-Indizes mit $\mathbf{l} = (l_1, \ldots, l_d) \in \mathbb{N}^d$, $\mathbf{i} = (i_1, \ldots, i_d) \in \mathbb{N}^d$. Dabei beschreibe $\mathbf{l}$ das Level, $\mathbf{i}$ den Index einer Größe in Abhängigkeit von der Dimension. Die d-dimensionalen, stückweise d-linearen Basisfunktionen $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$, $\mathbf{x} \in \Omega$, seien das Tensorprodukt der zugehörigen eindimensionalen Basisfunktionen $\phi_{l,i}(x)$ aus Gleichung 3.1:

$$\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) := \prod_{j=1}^{d} \phi_{l_j,i_j}(x_j) \tag{4.1}$$

Jede dieser Hutfunktionen ist zentriert auf dem Gitterpunkt

$$\mathbf{x}_{\mathbf{l},\mathbf{i}} := (x_{l_1,i_1}, x_{l_2,i_2}, \ldots, x_{l_d,i_d}),$$

mit $x_{l_j,i_j} := i_j \cdot h_{l_j} = i_j \cdot 2^{-l_j}$, und besitzt den Träger

$$\text{support}(\phi_{\mathbf{l},\mathbf{i}}) = \left[x_{l_1,i_1} - 2^{-l_1}, x_{l_1,i_1} + 2^{-l_1}\right] \times \cdots \times \left[x_{l_d,i_d} - 2^{-l_d}, x_{l_d,i_d} + 2^{-l_d}\right].$$

Der Raum der stückweise d-linearen Funktionen $V_\mathbf{l}$ mit Randwerten 0 und Maschenweiten $h_\mathbf{l} := (h_{l_1}, \ldots, h_{l_d}) = (2^{-l_1}, \ldots, 2^{-l_d})$ wird dann aufgespannt von den Basisfunktionen der Basis

$$\Phi_{V_\mathbf{l}} := \left\{\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) : \; i_j = 1, \ldots, 2^{l_j} - 1, \; j = 1, \ldots, d\right\}. \tag{4.2}$$

Das zugrunde liegende Gitter ist äquidistant bezüglich jeder Koordinatenrichtung und kann als kartesisches Produkt der eindimensionalen Gitter mit Level l_i in Richtung x_i aufgefasst werden. Für die Anzahl der Basisfunktionen gilt somit

$$|\Phi_{V_\mathbf{l}}| = \prod_{j=1}^{d} \left(2^{l_1} - 1\right). \tag{4.3}$$

Sei V_n der Raum der stückweise linearen Funktionen mit äquidistantem Gitter in jeder Richtung, d. h. $V_n := V_\mathbf{l}$ mit $n = l_1 = \cdots = l_d$. Bei gleichem Level l in jeder Dimension ist die Kardinalität der Basis $\mathcal{O}(2^{ld})$. Abbildung 4.1 zeigt die Gitterpunkte zu den zweidimensionalen Basen $\Phi_{V_\mathbf{l}}$ bis Level 3 in x_1- und x_2-Richtung. Die Träger der Basisfunktionen überlappen sich mit denen ihrer acht Nachbarn. An jedem Gitterpunkt ist nur die entsprechende Basisfunktion ungleich null. Die V_n mit $n \in \{1,2,3\}$ liegen auf der Diagonalen.

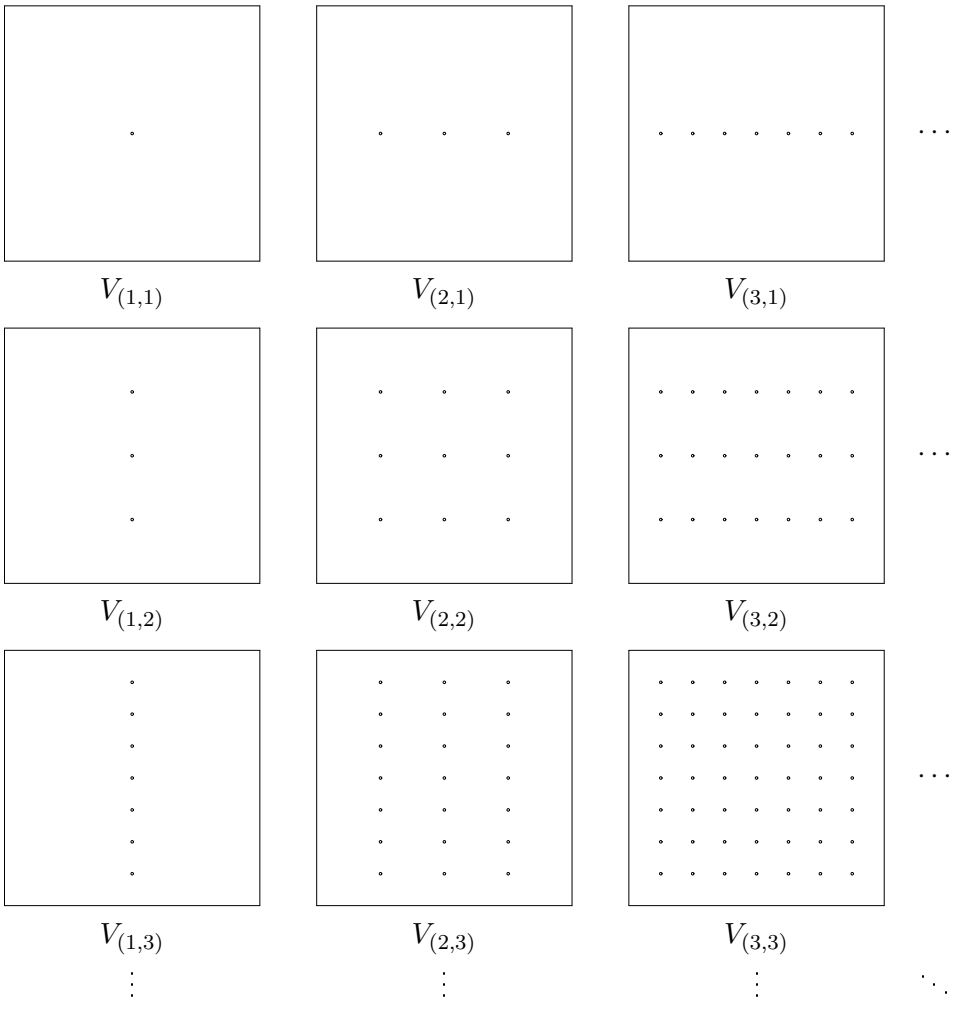


Abbildung 4.1: Gitterpunkte zu den Basen $\Phi_{V_\mathbf{l}}$ der Funktionsräume $V_\mathbf{l}$ für $l_j = 1, 2, 3$, $j = 1, 2$.

4.2 Hierarchische Basisfunktionen

Analog lassen sich für die hierarchische Basis die Teilräume $W_{\mathbf{l}}$ definieren:

$$W_{\mathbf{l}} := \operatorname{span}\left(\Phi_{\mathbf{l},\mathbf{i}}\right), \tag{4.4}$$

mit der Basis

$$\Phi_{W_{\mathbf{l}}} := \left\{\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) : \; i_j = 1, \ldots, 2^{l_j} - 1, \; i_j \text{ ungerade}, \; j = 1, \ldots, d\right\}. \tag{4.5}$$

Die Basisfunktionen einer Basis sind das Tensorprodukt der eindimensionalen Basisfunktionen der hierarchischen Basen auf den Level l_j. Abbildung 4.2 zeigt die Basisfunktionen der Räume $W_{\mathbf{l}}$ in zwei Dimensionen bis Level 3 in jeder Richtung. Die Abbildungen der Basen veranschaulichen die Entstehung der Basisfunktionen über das Tensorprodukt der eindimensionalen Versionen und zeigen, dass die Träger der Basisfunktionen eines Raumes paarweise disjunkt sind.

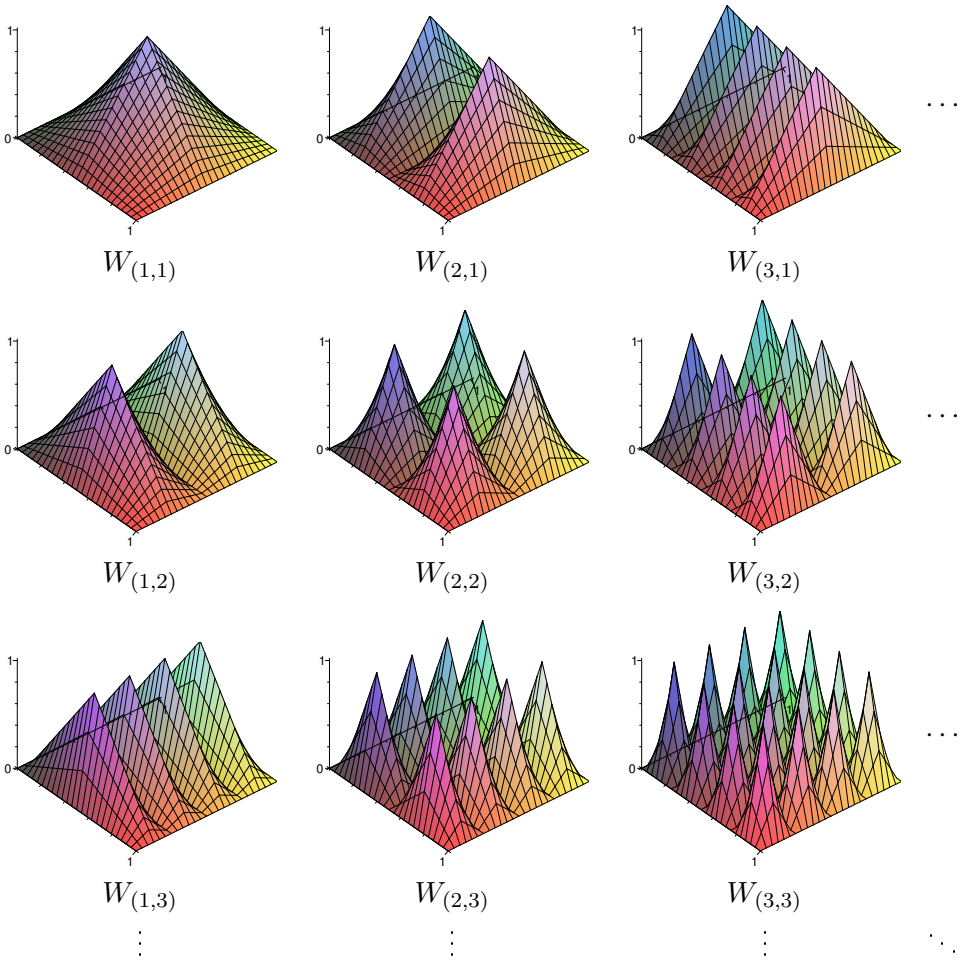


Abbildung 4.2: Die bilinearen Basisfunktionen der Räume $W_{\mathbf{l}}$ für $l_j = 1, 2, 3$, $j = 1, 2$.

Im Folgenden werden die diskreten L_1- und L_∞-Normen benötigt. Sei $\boldsymbol{\alpha}$ ein Multiindex der Länge d. Dann ist

$$|\boldsymbol{\alpha}|_1 := \sum_{j=1}^{d} \alpha_j$$

und

$$|\boldsymbol{\alpha}|_\infty := \max_{1 \le j \le d} \alpha_j.$$

Der Raum $V_n^{(\infty)} := V_n$ lässt sich darstellen als Summe von Teilräumen $W_{\mathbf{l}}$:

$$V_n^{(\infty)} := \sum_{l_1=1}^{n} \cdots \sum_{l_d=1}^{n} W_{(l_1,\ldots,l_d)} = \bigoplus_{|\mathbf{l}|_\infty \leq n} W_{\mathbf{l}} \tag{4.6}$$

Es ist

$$|W_{\mathbf{l}}| := \dim W_{\mathbf{l}} = \prod_{j=1}^{d} 2^{l_j - 1} = 2^{|\mathbf{l}|_1 - d}. \tag{4.7}$$

Die Summe ist direkt, d. h. für ein $f = \sum_{\mathbf{l}} f_{\mathbf{l}}$ mit $f_{\mathbf{l}} \in W_{\mathbf{l}}$ und paarweise verschiedenen Indizes $\mathbf{l}$ ist $f = 0$ genau dann, wenn alle $f_{\mathbf{l}} = 0$ sind.

Da die Anzahl der Basisfunktionen der Anzahl der Freiheitsgrade entspricht, muss gelten:

$$\left| V_n^{(\infty)} \right| = \bigoplus_{|\mathbf{l}|_\infty \leq n} |W_{\mathbf{l}}| \,. \tag{4.8}$$

Beweis:

$$\begin{aligned}
\bigoplus_{|\mathbf{l}|_\infty \leq n} |W_{\mathbf{l}}| &= \sum_{l_1=1}^{n} \cdots \sum_{l_d=1}^{n} 2^{(l_1-1)+\cdots+(l_d-1)} \\
&= \left(\sum_{l_1=1}^{n} 2^{l_1-1} \right) \cdots \left(\sum_{l_d=1}^{n} 2^{l_d-1} \right) \\
&= \left(\sum_{l_1=0}^{n-1} 2^{l_1} \right) \cdots \left(\sum_{l_d=0}^{n-1} 2^{l_d} \right) \\
&= (2^n - 1) \cdots (2^n - 1) \\
&= (2^n - 1)^d \\
&= |V_n|
\end{aligned}$$

Im Vergleich von Abbildung 4.1 mit Abbildung 4.3 wird dies für $V_3^{(\infty)} = V_{(3,3)}$ anschaulich klar. Die Anzahl der Freiheitsgrade ist hier $(2^3 - 1)^2 = 49$, die Summe aller Gitterpunkte der $W_{\mathbf{l}}$ im Tableau in Abbildung 4.3 ebenso.

Jede Funktion $f_n^{(\infty)} \in V_n^{(\infty)}$ kann analog aufgeteilt werden in

$$f_n^{(\infty)}(\mathbf{x}) = \sum_{|\mathbf{l}|_\infty \leq n} f_{\mathbf{l}}(\mathbf{x}), \text{ mit } f_{\mathbf{l}} \in W_{\mathbf{l}} \text{ und } f_{\mathbf{l}}(\mathbf{x}) = \sum_{\mathbf{i}: \phi_{\mathbf{l},\mathbf{i}} \in \Phi_{W_{\mathbf{l}}}} \alpha_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}). \tag{4.9}$$

Die hierarchische Basis kann daher verwendet werden, um das LGS zur Klassifikation in Gleichung 2.13 aufzustellen. Es müssen lediglich die Basisfunktionen sowie die zugehörigen Koeffizienten geeignet aufgefädelt werden, um sie in eine feste Reihenfolge zu bringen. Die Basisfunktionen werden benötigt, um die Matrix C aufzustellen, und, abhängig von den Trainingsdaten, um die Matrix B aufzustellen. Das Trainieren des Systems (das Lösen des Gleichungssystems) liefert Werte für alle hierarchischen Überschüsse $\alpha_{\mathbf{l},\mathbf{i}}$. Mit diesen kann die gelernte Funktion nach Gleichung 4.9 aufgestellt werden. Allerdings ist die Anzahl der Gitterpunkte oder Freiheitsgrade immer noch $\mathcal{O}(2^{nd})$.

Betrachte nun die Interpolation der Funktion $f(\mathbf{x}) := 4x_1(1 - x_1) \cdot 4x_2(1 - x_2)$, die das Tensorprodukt der bereits in Kapitel 3 vorgestellten Parabel jeweils in x_1- und x_2-Richtung ist, siehe Abbildung 4.4. Eingezeichnet sind die Gitterpunkte für die hierarchischen Basisfunktionen von $V_3^{(\infty)}$, den Raum der stückweise bilinearen Funktionen mit Maschenweite 1/8 in jeder Richtung.

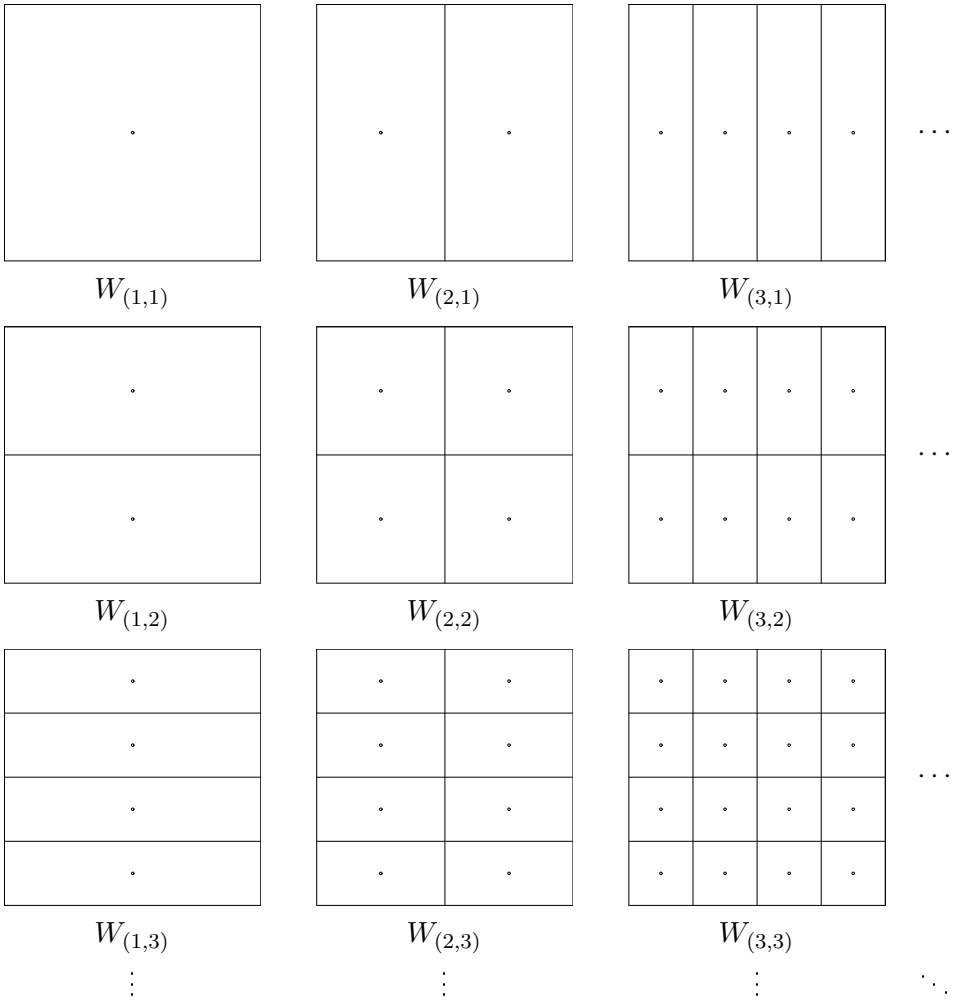


Abbildung 4.3: Gitterpunkte zu den Basen $W_{\mathbf{l}}$ für $l_j = 1, 2, 3$, $j = 1, 2$. Die Linien zwischen den Gitterpunkten sind die Grenzen der Träger der zugehörigen Basisfunktionen. Die Träger aller Basisfunktionen eines Raumes $W_{\mathbf{l}}$ sind paarweise disjunkt und überdecken zusammen ganz Ω.

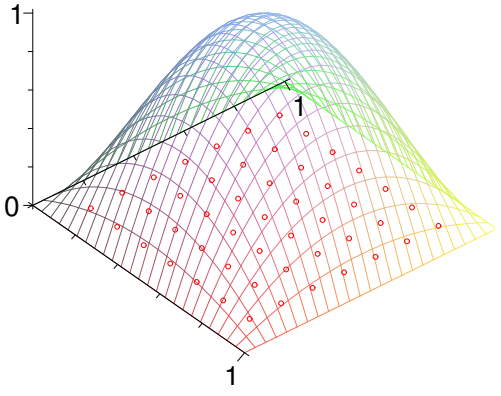


Abbildung 4.4: $f(\mathbf{x})$ mit Gitterpunkten

$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{16}$	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$
$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{4}$	$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{32}$
$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{16}$	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$
$\frac{1}{16}$	$\frac{1}{4}$	$\frac{1}{16}$	1	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{1}{16}$
$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{16}$	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$
$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{4}$	$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{32}$
$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{16}$	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{64}$

Tabelle 4.1: Hierarchische Überschüsse

Tabelle 4.1 enthält die hierarchischen Überschüsse. Die mittlere Zeile und die mittlere Spalte erinnern an den eindimensionalen Fall, vergleiche Tabelle 3.2. Dort wurde bereits erwähnt, dass die Größenordnung des Fehlers bei Interpolation mit einer stückweise linearen Funktion $\mathcal{O}(h^2)$ ist. Tatsächlich kann für den allgemeinen Fall gezeigt werden [BuGr04], dass für den Beitrag der $f_{\mathbf{l}}$ aus den Teilräumen $W_{\mathbf{l}}$ zur Funktion $f \in V_n^{(\infty)}$ in der L_2-Norm gemessen gilt:

$$\|f_{\mathbf{l}}\|_{L_2} \leq 3^{-d} \cdot 2^{-2\cdot|\mathbf{l}|_1} \cdot |f|_{\mathbf{2},2}, \tag{4.10}$$

wobei die Halbnorm $|f|_{\mathbf{2},2}$ ein Maß für die zweiten Ableitungen von f ist, vgl. [BuGr04].

Interessant ist hierbei, dass der Beitrag der $f_{\mathbf{l}}$ exponentiell in $|\mathbf{l}|_1$ abnimmt. In einer systematischen Auflistung der Teilräume wie in Abbildung 4.3 bedeutet dies anschaulich, dass, je weiter man sich in der Manhattan-Distanz von $W_{(1,\ldots,1)}$ von Teilraum zu Teilraum wegbewegt, desto geringer wird der Beitrag zur gesuchten Funktion f. Diese Betrachtung und ein Blick auf die Koeffizienten in Tabelle 4.1 legen nahe, ab einem bestimmten Wert für $|\mathbf{l}|_1$ „abzuschneiden“, und führen direkt zur Definition des Dünngitterraumes $V_n^{(1)}$:

$$V_n^{(1)} := \bigoplus_{|\mathbf{l}|_1 \leq n+d-1} W_{\mathbf{l}} \tag{4.11}$$

Im Tableau der Teilräume in Abbildung 4.3 würde dies bedeuten, nur alle Teilräume bis inklusive der Geraden $l_1 + l_2 = n+1$ zu verwenden. Im Falle der Interpolation der Funktion $f(\mathbf{x}) := 4x_1(1-x_1) \cdot 4x_2(1-x_2)$ mit einer Funktion $f \in V_3^{(1)}$ würden alle Basisfunktionen wegfallen, deren Beitrag kleiner als $1/16$ ist. Die Gitterpunkte des dünnen Gitters zu $n = 3$ sind in Abbildung 4.5, die übrig gebliebenen Überschüsse in Tabelle 4.2 dargestellt.

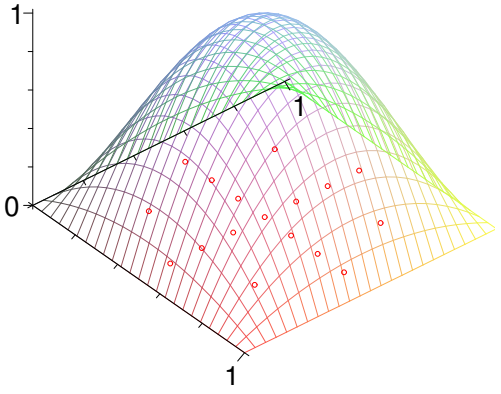


Abbildung 4.5: $f(\mathbf{x})$ mit Gitterpunkten

		$\frac{1}{16}$				
	$\frac{1}{16}$		$\frac{1}{4}$		$\frac{1}{16}$	
		$\frac{1}{16}$				
$\frac{1}{16}$	$\frac{1}{4}$	$\frac{1}{16}$	1	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{1}{16}$
		$\frac{1}{16}$				
	$\frac{1}{16}$		$\frac{1}{4}$		$\frac{1}{16}$	
		$\frac{1}{16}$				

Tabelle 4.2: Hierarchische Überschüsse

Wiederum kann jede Funktion $f_n^{(1)} \in V_n^{(1)}$ als Summe der Funktionen der Teilräume dargestellt werden:

$$f_n^{(1)}(\mathbf{x}) = \sum_{|\mathbf{l}|_1 \leq n+d-1} f_{\mathbf{l}}(\mathbf{x}), \text{ mit } f_{\mathbf{l}} \in W_{\mathbf{l}}. \tag{4.12}$$

Die Anzahl der Freiheitsgrade oder Gitterpunkte in $V_n^{(1)}$ ist nun

$$|V_n^{(1)}| = \bigoplus_{|\mathbf{l}|_1 \leq n+d-1} |W_{\mathbf{l}}| \in \mathcal{O}(2^n \cdot n^{d-1}). \tag{4.13}$$

Die Größenordnung des Fehlers ist

$$\|f - f_n^{(1)}\|_{L_2} \in \mathcal{O}(h_n^2 \cdot \log(h_n^{-1})^{d-1}) = \mathcal{O}(2^{-2n} \cdot n^{d-1}), \tag{4.14}$$

unter der Voraussetzung, dass geringfügig stärkere Glattheitsbedingungen an f gestellt werden, was im Fall der Klassifikation keine Einschränkung bedeutet: Der Klassifikator muss für die gewünschte Generalisierungsfähigkeit ohnehin einen gewissen Grad der Glattheit besitzen. Für Herleitung und Beweise der Zahl der Gitterpunkte und der Konvergenzordnung siehe [BuGr04].

Im Vergleich zum vollständigen Gitter mit $\mathcal{O}(2^{nd})$ Gitterpunkten ist dies eine deutliche Reduktion der Anzahl der Freiheitsgrade, während die Größenordnung des Fehlers bei der Interpolation ($\mathcal{O}(h_n^2)$) bis auf logarithmische Terme gleich bleibt. Abbildung 4.2 zeigt ein Dünnes Gitter in zwei und drei Dimensionen für Level 6 in jeder Dimension.

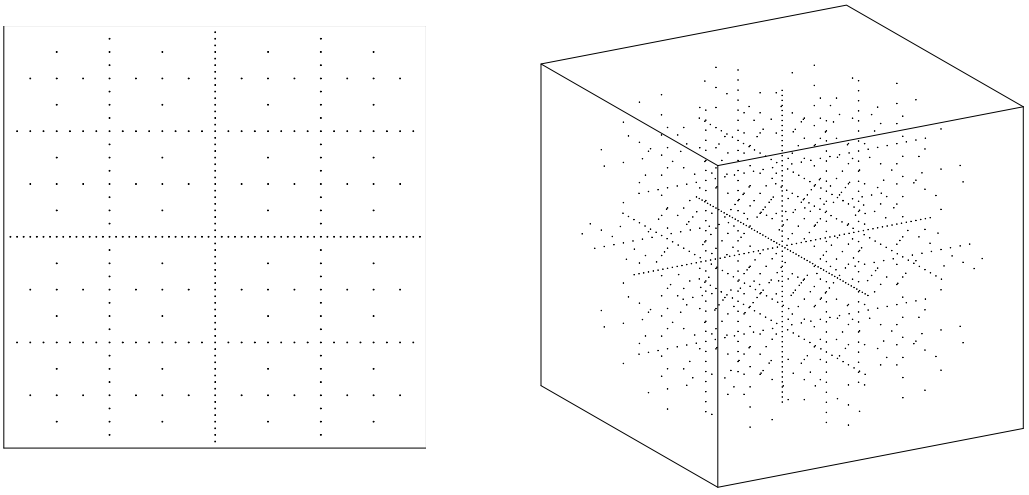


Abbildung 4.6: Dünnes Gitter zu $V_n^{(1)}$ für $n = 6$ in zwei und drei Dimensionen.

4.3 Randbetrachtung

Die Berücksichtigung von Randwerten kann dank der Tensorproduktkonstruktion der Basisfunktionen im Höherdimensionalen analog zum eindimensionalen Fall erfolgen.

Eine Möglichkeit ist wieder, zusätzliche Basisfunktionen mit Gitterpunkt auf dem Rand zu verwenden. Im Eindimensionalen Fall sind dies die Hutfunktionen $\phi_{l,0}$ und $\phi_{l,2^l}$ (Knotenbasis) bzw. $\phi_{0,0}$ und $\phi_{0,1}$ (hierarchische Basis), siehe dazu auch Kapitel 3.3. Bei d Dimensionen hat dies allerdings für die Basis des $V_n^{(1)}$ zur Folge, dass etliche neue Basisfunktionen hinzukommen, die auf dem Rand von Ω liegen.

Alternativ kann die Basis mit ausgeklappten Randfunktionen aus dem Eindimensionalen auf d Dimensionen erweitert werden. Abbildung 4.7 zeigt die Basisfunktionen für die modifizierten Teilräume $W_{\mathbf{l}}$ in zwei Dimensionen. Im Gegensatz zu den $V_n^{(\infty)}$ aus dem letzten Kapitel, den Räumen der stückweise d-linearen Funktionen mit Maschenweite $h = 2^{-n}$, geht eine Verfeinerungsstufe am Rand verloren. Die Maschenweite h bleibt im Teilraum $[2h, 1-2h]^d$ zwar erhalten; am Rand ergibt sich jedoch eine Maschenweite von $2h$. $W_{(1,1)}$ ist nun der Raum der konstanten Funktionen, $V_2^{(\infty)}$ der Raum der stückweise bilinearen Funktionen mit Maschenweite $2h = 1/2$ (und Randwerten nicht notwendigerweise null).

Soll die konstante Maschenweite in jeder Richtung erhalten bleiben, so können die Daten auf $[2^{-l_1}, 1-2^{-l_1}] \times \cdots \times [2^{-l_d}, 1-2^{-l_d}]$ normalisiert werden oder bei der Tensorproduktbildung der Basis in jeder Dimension die Randbasisfunktionen des nächst höheren Levels hinzugenommen werden. Zweiteres würde allerdings wiederum zusätzliche Basisfunktionen bedeuten und der Verwendung von Basisfunktionen auf dem Rand entsprechen.

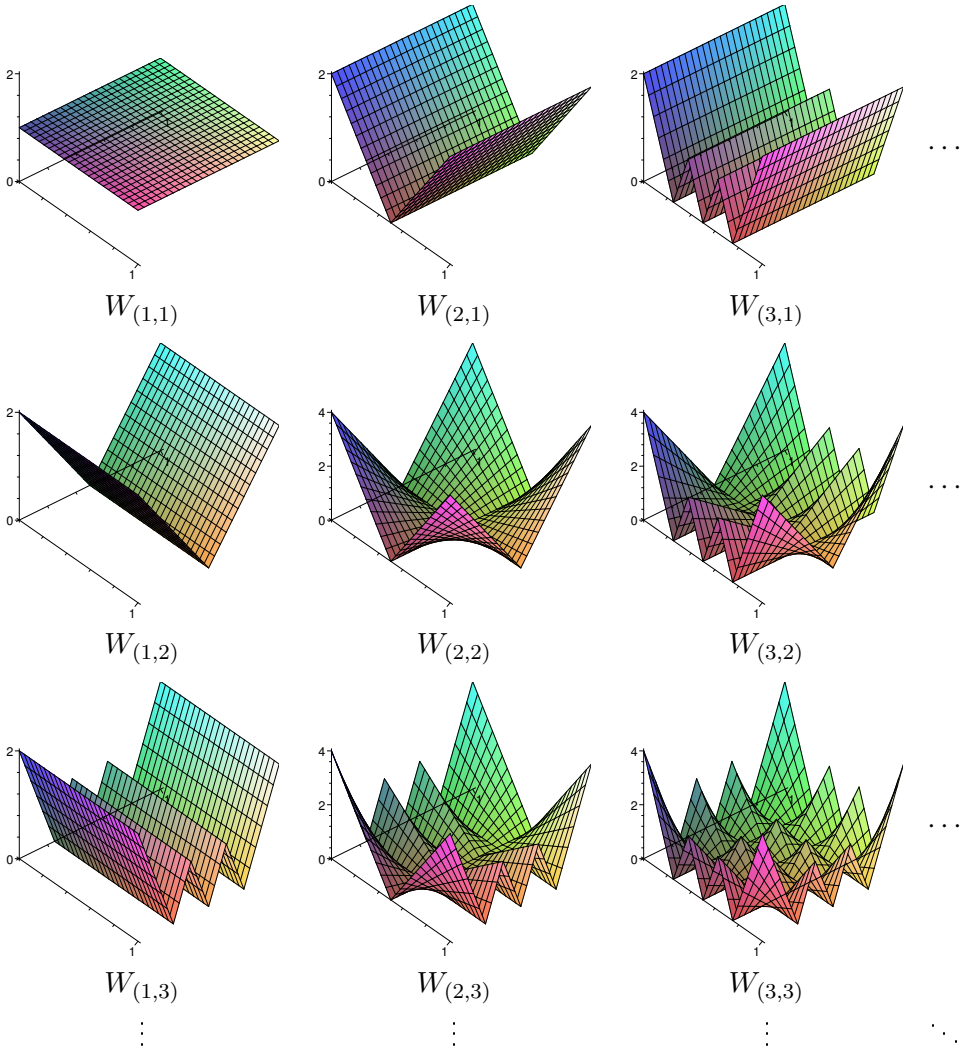


Abbildung 4.7: Die modifizierten Basisfunktionen der Räume $W_{\mathbf{l}}$ für $l_j = 1, 2, 3$, $j = 1, 2$. Maschenweite des aufgespannten Raumes der stückweise bilinearen Funktionen ist am Rand $2h$.

4.4 Kombinationstechnik

Durch die Verwendung der Dünngittertechnik kann die Größe des Problems von $\mathcal{O}(2^{nd})$ (bei Verwendung eines vollen Gitters) auf $\mathcal{O}(2^n \cdot n^{d-1})$ reduziert werden – bei vergleichbarer Approximationsgenauigkeit. Dennoch erreichen die Matrizen des diskreten Systems (2.13) schnell unhandliche Größen. Ein explizites Aufstellen sollte vermieden werden. In iterativen Lösern wie z. B. dem Verfahren der Konjugierten Gradienten sollten daher im Wesentlichen nur die Anwendungen der Matrizen, die Matrix-Vektor-Multiplikationen, implementiert werden:

$$\left(\lambda C_{\mathbf{l}} + B_{\mathbf{l}} \cdot B_{\mathbf{l}}^T\right) \boldsymbol{\alpha} = \lambda C_{\mathbf{l}} \cdot \boldsymbol{\alpha} + B_{\mathbf{l}} \cdot (B_{\mathbf{l}}^T \cdot \boldsymbol{\alpha}). \tag{4.15}$$

In [GaGT01] wird stattdessen die Dünngitter-Kombinationstechnik [GrSZ92] zur Klassifikation verwendet. Hierbei wird der Merkmalsraum für eine gewisse Anzahl vollbesetzter Gitter mit einheitlicher Maschenweite je Koordinatenrichtung diskretisiert. Die entstehenden diskreten Systeme werden einzeln gelöst und die Ergebnisse mittels multivariater Extrapolation geeignet zusammengeführt. Betrachtet werden die Funktionsräume

$$V_{\mathbf{l}},\ \mathbf{l} \in I_n^{(\mathrm{C})} \tag{4.16}$$

mit

$$I_n^{(\mathrm{C})} := \{\mathbf{l} :\ |\mathbf{l}|_1 = (n+d-1) - q,\ q = 0, \ldots, d-1\}. \tag{4.17}$$

Um Randwerte ungleich null zuzulassen, seien die zugehörigen Basen für die Betrachtung der Kombinationstechnik ergänzt um Basisfunktionen mit Gitterpunkt auf dem Rand, vergleiche Kapitel 3.3 und 4.3. Abbildung 4.8 zeigt die verwendeten Gitter für zwei Dimensionen und $n = 4$.

Zu lösen sind die Systeme

$$\left(\lambda C_{\mathbf{l}} + B_{\mathbf{l}} \cdot B_{\mathbf{l}}^T\right) \boldsymbol{\alpha}_{\mathbf{l}} = B_{\mathbf{l}}\, y,\ \ \mathbf{l} \in I_n^{(\mathrm{C})}. \tag{4.18}$$

Aus deren Lösungen,

$$f_{\mathbf{l}}(\mathbf{x}),\ \ \mathbf{l} \in I_n^{(\mathrm{C})}, \tag{4.19}$$

wird die resultierende Funktion $f_n^{(\mathrm{C})}$ durch Linearkombination wie folgt gewonnen:

$$f_n^{(\mathrm{C})}(\mathbf{x}) := \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{|\mathbf{l}|_1 = (n+d-1)-q} f_{\mathbf{l}}(\mathbf{x}). \tag{4.20}$$

Die Lösung $f_n^{(\mathrm{C})}$ ist im Allgemeinen nicht die selbe wie die direkte Lösung des Gleichungssystems 2.13, $f_n^{(1)}$. Die Genauigkeit ist jedoch für viele Probleme von der selben Größenordnung. Für weitere Details hierzu, siehe beispielsweise [GrSZ92].

Anstelle von einem Problem der Größe $\dim(V_n^{(1)}) \in \mathcal{O}(2^n \cdot n^{d-1})$ sind nun $\mathcal{O}(d \cdot n^{d-1})$ Probleme der Größe $\dim(V_{\mathbf{l}}) \in \mathcal{O}(2^n)$ zu lösen. Die Problemgröße des Einzelproblems ist geringer und lässt bei gleichen Einschränkungen z. B. an die Größe des zur Verfügung stehenden Speichers das Berechnen größerer Systeme zu als das direkte Lösen des Systems 2.13. Zudem sind äquidistante Gitter einfacher zu handhaben und es stehen effiziente Löser zur Verfügung. Ein weiterer Vorteil ist oft die einfache Parallelisierbarkeit, da die Teilprobleme unabhängig voneinander gelöst werden können; allerdings stehen Parallelrechner meist bei Klassifikationsanwendungen nicht zur Verfügung.

Im Folgenden wird die Realisierung einer Testumgebung mittels dünner Gitter vorgestellt. Diese Testumgebung ermöglicht sowohl die Verwendung der direkten Klassifikation mittels dünner Gitter, als auch die Verwendung der Kombinationstechnik. Anschließend wird das System für beide Techniken auf Testbeispiele angewendet.

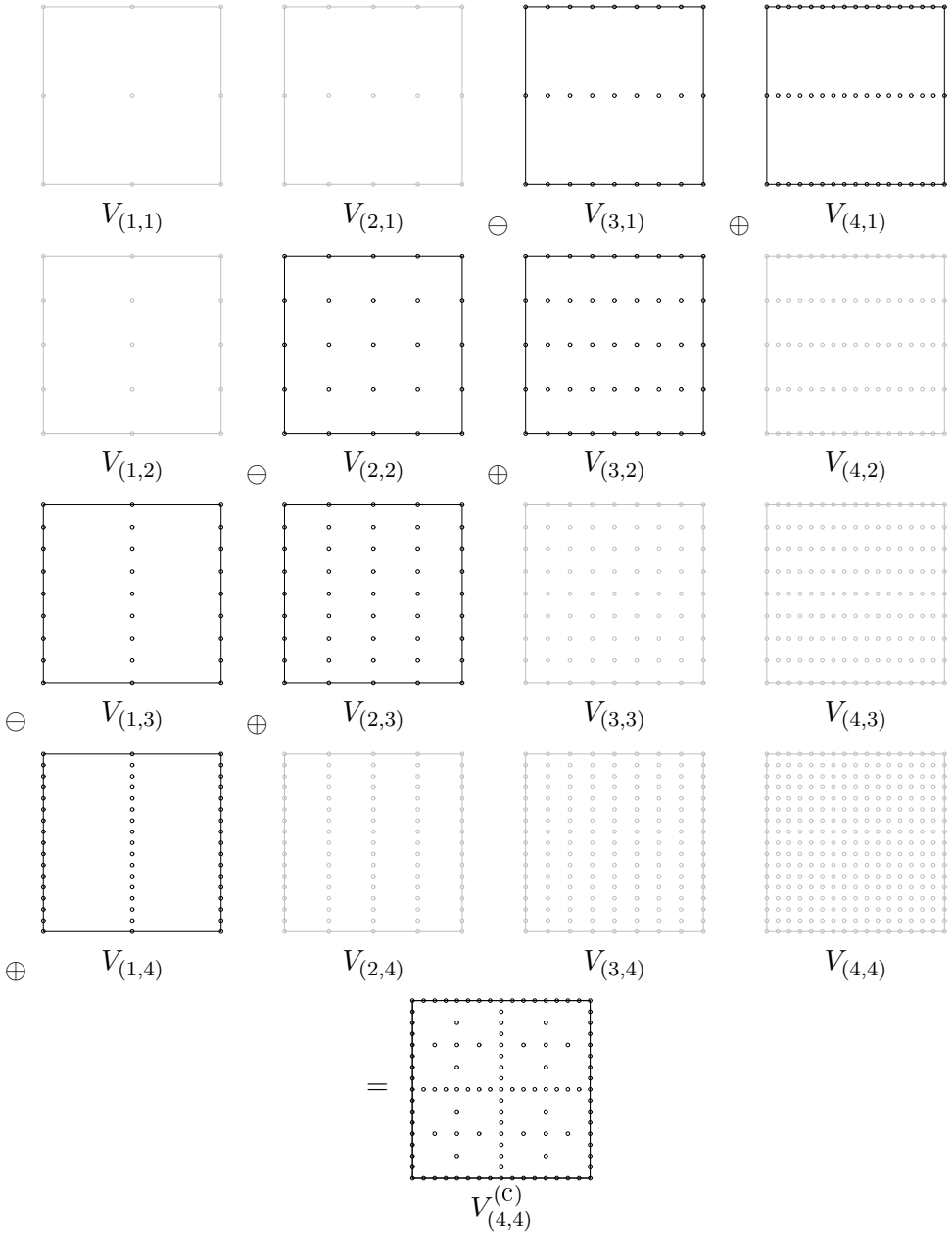


Abbildung 4.8: Kombinationstechnik für $n = 4$, $d = 2$. Lösungen $f_{\mathbf{l}}$ der V_{l_1,l_2} für $l_1 + l_2 = n + 1$ werden addiert, für $l_1 + l_2 = n$ subtrahiert.

5 Realisierung

Kernstück dieser Arbeit war die Erstellung eines Systems zur Klassifikation mit Hilfe der Dünngittertechnik. Dieses System sollte insbesondere flexibel sein, um als Test- und Vergleichsplattform verschiedener Techniken dienen zu können. Die Abwägung zwischen Effizienz und Flexibilität fiel zu Gunsten der Flexibilität aus.

Die Wahl der Programmierumgebung fiel auf Maple. Da Maple als mathematisches Computeralgebra-System nicht nur effiziente mathematische Datentypen und Operationen besitzt, sondern auch symbolisches Rechnen beherrscht, eignet es sich besonders, um ein System zu realisieren, in dem auf einfache Art und Weise verschiedene Ansätze getestet werden können.

Entstanden ist ein Maple-Paket, `DG` (für Dünne Gitter), das einfach eingebunden werden kann und Klassifikationswerkzeuge zur Verfügung stellt.

5.1 Vorbereiten der Daten

Die Trainings- und Testdaten müssen in einem von Maple lesbaren Format vorliegen, getrennt nach Datum und Klassifikation. Das Paket `DG` benötigt die Datenpunkte als $M \times d$-Matrix vom Typ `Matrix` des Pakets `LinearAlgebra` und die zugehörigen Klassen in einem Vektor mit M Einträgen vom Typ `Vector`. Im Folgenden wird eine Datenmatrix mit `X` und der zugehörige Klassenvektor mit `Y` bezeichnet.

Als Vorverarbeitungsschritt müssen die Datenpunkte auf $[0,1]^d$ normalisiert werden und die Klassenlabel auf zwei reelle Werte, die betragsmäßig gleich und vom Vorzeichen her unterschiedlich sind.

`DG` stellt hierfür zwei Prozeduren bereit, `normalize_data` und `normalize_class`. Die Prozedur `normalize_data` normalisiert die Einträge in einer Matrix `X` auf $[0,1]$ je Dimension, `normalize_class` die Klassenlabel eines Vektors `Y` auf $+1$ und -1.

5.2 Basis und Gitterpunkte

Voraussetzungen für das Aufstellen des linearen Gleichungssystems sind die Definition der (hierarchischen) Basisfunktionen und die Erstellung eines Gitters. Die Basisfunktion muss als Maple-Funktion in Abhängigkeit von Level und Index vorliegen und als Rückgabe die Basisfunktion in einer Dimension liefern. Für die Hutfunktion zur hierarchischen Basis entspricht dies Gleichung 3.1, oder, in Maple-Syntax,

```
[> (level,index) ->
        ( x -> piecewise(x <= (index-1)/2^level, 0,
                         x <= index/2^level, 2^level*x+1-index,
                         x <= (index+1)/2^level, index+1-2^level*x) );
```

Sämtliche Gitter müssen für die Verwendung in `DG` als Liste von Listen mit jeweils Level und Index pro Dimension vorliegen, also in der Form

$$[[l_1, i_1, \ldots, l_d, i_d], \ldots, [l_1, i_1, \ldots, l_d, i_d]],$$

wobei jede Liste einem Gitterpunkt mit der zugehörigen Basisfunktion $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \phi_{l_1,i_1}(x_1) \cdot \phi_{l_2,i_2}(x_2) \cdot \ldots \cdot \phi_{l_d,i_d}(x_d)$ entspricht.

Im Paket `DG` kann zwischen zwei vorgefertigten Realisierungen gewählt werden. Die Auswahl kann mit den Befehlen

```
[> set_basis("hierarchisch_h");
```

und

```
[> set_basis("hierarchisch_2h");
```

getroffen werden. Beide Varianten verwenden die in Kapitel 4.3 besprochene Basis mit ausgeklappten Randbasisfunktionen: Die erste Variante mit Basisfunktionen zu Gitterpunkten auf dem Rand, die zweite ohne.

5.2.1 Erzeugung der Gitterpunkte

Die Erzeugung der Liste der Gitterpunkte kann auf verschiedene Arten erfolgen. Die Realisierung in Listing 5.1 orientiert sich an dem Tableauschema der Abbildungen 4.3 und 4.7. Startpunkt ist das Gitter zum Dünngitterraum $V_1^{(1)}$. Begonnen wird in Dimension d auf Level $l_d = 1$. Bis die Abbruchbedingung $|\mathbf{l}|_1 = n + d - 1$ erfüllt ist, werden rekursiv zum einen die Teilräume in der selben Dimension bis zum maximal erreichbaren Level abgearbeitet. Zum anderen wird das Level in der aktuellen Dimension fixiert und die Gitter zu den Teilräumen in den anderen $d - 1$ Dimensionen bis zum noch übrig gebliebenen Restlevel werden erstellt. Beide Teilmengen der Gitterpunkte werden vereinigt (Konkatenation der beiden Teillisten). Bei Abstieg in der Dimension werden über das kartesische Produkt des eindimensionalen Gitters auf dem aktuellen Level und der Gitterpunkte in $d - 1$ Dimensionen Gitterpunkte in d Dimensionen erzeugt.

Eine weitere alternative Möglichkeit zur rekursiven Erzeugung eines dünnen Gitters basiert auf folgendem Ansatz: Das Gitter zum Raum $V_n^{(1)}$ in d Dimensionen kann zusammengesetzt werden aus einem Gitter zum Raum $V_n^{(1)}$ mit der um eins kleineren Dimensionalität $d - 1$ und zwei Gittern zum ebenfalls d-dimensionalen Raum aber mit dem um eins reduzierten Level $n-1$, $V_{n-1}^{(1)}$. Abbildung 5.1 zeigt eine anschauliche Darstellung des Rekursionsansatzes für das dünne Gitter zum Ansatzraum $V_3^{(1)}$ in zwei Dimensionen.

Auf welche Art und Weise die Liste der Gitterpunkte erzeugt wird, ist nicht von Bedeutung; das dünne Gitter lässt sich beispielsweise ebenso iterativ erzeugen. Im Hinblick auf Optimierungen ist es allerdings hilfreich, wenn zumindest die Gitterpunkte des eindimensionalen Gitters absteigend nach Größe des Trägers, d. h. aufsteigend nach Level, geordnet sind.

Mit `set_basis` können eigene Funktionen zur Erstellung von Basisfunktionen (in Abhängigkeit von Level und Index in einer Dimension) und der Liste der Gitterpunkte (in Abhängigkeit von dem Level n) gesetzt und bei der Klassifikation verwendet werden. Mit `get_phi_li()` lässt sich auf die aktuelle Basisfunktion, mit `get_li_liste()` auf die Funktion zur Erstellung der Liste der Gitterpunkte zugreifen. Es ist daher möglich, auch ein vollbesetztes Gitter für Vergleichsrechnungen zum Aufstellen des Gleichungssystems zu verwenden.

```
li_liste_hb2h := proc(dim::integer, level::integer)
  # dim   ... Dimension des Merkmalraums
  # level ... Rekursionstiefe für alle Richtungen
  local eval_ml_rec;

  # eigentliche, rekursive Prozedur, die die Liste erstellt
  eval_ml_rec := proc(dim::integer, akt_level::integer, level::integer)
    local basisfunktionen;

    # Liste der Basisfunktionen auf aktuellem Level,
    # hierarchische Basis
    basisfunktionen := [seq([akt_level, 2*ind-1], ind=1..2^(akt_level-1))];

    # Abbruchkriterium
    if (dim = 1) and (akt_level = level) then
      return basisfunktionen;
    # 1D-Fall: alle übrigen Level ablaufen
    elif (dim = 1) then
      return [op(basisfunktionen), op(eval_ml_rec(dim, akt_level+1, level))];
    # nD-Fall, aktuelles Level erschöpft:
    elif (akt_level = level) then
      return cart_product(basisfunktionen,
                          eval_ml_rec(dim-1, 1, level-akt_level+1));
    # allgemeiner Fall, 1) noch Dimensionen übrig,
    #                   2) noch Level in aktueller Dimension übrig
    else
      return [op(cart_product(basisfunktionen,
                              eval_ml_rec(dim-1, 1, level-akt_level+1))),
              op(eval_ml_rec(dim, akt_level+1, level))];
    end if;
  end proc:

  # Aufruf der rekursiven Prozedur
  return eval_ml_rec(dim, 1, level);
end proc:
```

Listing 5.1: Erstellung der Gitterpunkte eines dünnen Gitters für `dim` Dimensionen und Level `level`. Variante ohne zusätzliche Punkte in Randnähe.

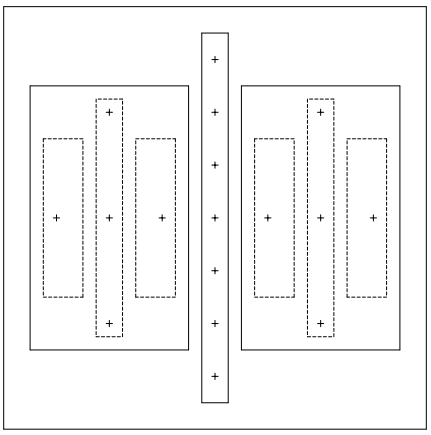


Abbildung 5.1: Rekursive Aufteilung des dünnen Gitters für $n = 3$, $d = 2$.

5.3 Klassifikation

Für die Klassifikation sind sowohl die direkte Dünngittertechnik als auch die Kombinationstechnik (Kap. 4.4) implementiert worden. Mit

```
[> classify(level, lambda, X, Y);
```

wird die direkte, mit

```
[> classify_combi(level, lambda, X, Y);
```

wird die Kombinationsmethode verwendet. Als Parameter werden neben den Trainingsdaten `X` und den zugehörigen Klassifikationen `Y` der Regularisierungsparameter `lambda` und das Level `level` benötigt. Die Dimension des Merkmalraumes ergibt sich aus der Anzahl der Attribute der Trainingsdaten, d. h. der Spalten der Matrix `X`. Die Prozedur zur Erstellung des Gitters wird mit Level und Dimension aufgerufen und das Gleichungssystem – bzw. im Fall der Kombinationstechnik die Gleichungssysteme – explizit aufgestellt und direkt gelöst, was die Verwendung beliebiger Ansatzräume ermöglicht.

Rückgabe der Klassifikation ist insbesondere die gelernte Funktion als Maple-Funktion, bei der Kombinationstechnik ist dies die gewichtete Summe der Funktionen auf den einzelnen Gittern. Diese kann in Maple direkt zur Visualisierung oder Auswertung auf neue Trainingsdaten verwendet werden.

5.3.1 Effizientes Lösen des Systems

In Kapitel 4.4 wurde bereits erwähnt, dass das explizite Aufstellen der Matrizen C und B aus Gleichung 2.13 aus Gründen der Effizienz vermieden werden sollte. Im Folgenden soll die effiziente Lösung des Gleichungssystems für den Ansatzraum der stückweise linearen Funktionen diskutiert und eine konkrete Realisierung vorgestellt werden.

Ausgangspunkt der folgenden Betrachtungen ist, dass ein iteratives Lösungsverfahren, beispielsweise das der Konjugierten Gradienten, zur Lösung des Systems

$$\begin{array}{ccccc} \underbrace{\left(\lambda C + B \cdot B^T\right)} & \boldsymbol{\alpha} & = & \underbrace{By} & \\ A & \boldsymbol{\alpha} & = & b & \end{array}$$

verwendet wird. Iterative Verfahren beginnen mit einer Startlösung $\boldsymbol{\alpha}^{(0)}$ und berechnen daraus eine Folge von Näherungswerten $\boldsymbol{\alpha}^{(j)},\ j = 1, 2, \ldots$. Im Konvergenzfall strebt diese für $j \to \infty$ gegen die exakte Lösung $\boldsymbol{\alpha}$.

Wirklich teuer in jedem Iterationsschritt sind Matrix-Vektor-Produkte der Form $A \cdot \boldsymbol{\alpha}$. Daher soll die Multiplikation der Matrix A mit einem beliebigen Vektor $\boldsymbol{\alpha}$ genauer untersucht werden. Diese lässt sich schreiben als Summe der beiden Teilprobleme

$$\lambda C \cdot \boldsymbol{\alpha} \quad \text{und} \quad B \cdot (B^T \cdot \boldsymbol{\alpha}).$$

Multiplikation mit der Matrix B

Das Produkt $B^T \cdot \boldsymbol{\alpha}$ entspricht der Auswertung des Klassifikators in den Trainingsdaten $\mathbf{x}_1, \ldots, \mathbf{x}_M$: In einer Zeile j der Matrix B^T stehen die Werte der N Basisfunktionen $\phi_{\mathbf{l},\mathbf{i}}$ ausgewertet an der Stelle $\mathbf{x}_j$. Da durch den hierarchischen Aufbau der Dünngitterbasis die Träger der Basisfunktionen in den einzelnen Teilräumen $W_{\mathbf{l}},\ |\mathbf{l}|_1 \leq n + d - 1$, paarweise disjunkt sind, kann nur eine Basisfunktion pro Teilraum einen Wert ungleich null liefern.

Es genügt, für jeden Teilraum die betroffene Funktion zu identifizieren. Dies ist in einem Raum $W_{\mathbf{l}}$ mit einer einfachen Indexrechnung pro Koordinatenrichtung j möglich: $i_j = \lceil x_j * 2^{l_j - 1} \rceil \cdot 2 - 1$. Punkte auf den Intervallenden können ohne weiteres dem linken Intervall zugeschlagen werden, da die Ansatzfunktionen dort in der Regel null sind – nur der Wert $x_j = 0$ muss gesondert behandelt werden. Die Auswertung der Basisfunktion $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}_j)$ beschränkt sich auf das Produkt von d Auswertungen eindimensionaler Basisfunktionen:

$$\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}_j) = \prod_{k=1}^{d} \phi_{l_k,i_k}(x_{j_k}). \tag{5.1}$$

Für jede Basisfunktion $\phi_{\mathbf{l},\mathbf{i}}$ muss auf den zugehörigen Koeffizienten $\alpha_{\mathbf{l},\mathbf{i}}$ zugegriffen werden. Um einen effizienten Zugriff in konstanter Zeit zu ermöglichen, muss darauf verzichtet werden, den Vektor $\boldsymbol{\alpha}$ zu durchsuchen. Denkbar wäre, die Koeffizienten in einer Reihenfolge abzulegen, so dass der Index im Vektor direkt berechnet werden kann. Dies würde jedoch die Flexibilität des Systems einschränken. Eine Alternative ist das einmalige Anlegen einer Hashfunktion, die in konstanter Zeit zu gegebenen $\mathbf{l}$ und $\mathbf{i}$ den Index von $\alpha_{\mathbf{l},\mathbf{i}}$ im Vektor $\boldsymbol{\alpha}$ zurückliefert. Diese Variante wurde realisiert, siehe hierzu auch Kapitel 5.3.5.

Insgesamt ergibt dies für den Aufwand der Multiplikation der Matrix B mit einem Vektor $\boldsymbol{\alpha}$: In jedem der $\mathcal{O}(n^d)$ Teilräume müssen $\mathcal{O}(d)$ Indexrechnungen, Funktionsauswertungen und Multiplikationen erbracht werden – das Durchlaufen der Teilräume kann analog zur der bereits vorgestellten Konstruktion der Liste der Gitterpunkte geschehen. Dies muss für jedes der M Trainingsdaten durchgeführt werden, was einen Gesamtaufwand von $\mathcal{O}(M \cdot d \cdot n^d)$ ergibt.

Die Multiplikationen der Matrix B mit dem Ergebnisvektor $\boldsymbol{\alpha}' := B^T \cdot \boldsymbol{\alpha}$ und dem Klassenvektor $\mathbf{y}$ auf der rechten Seite des Gleichungssystems geschehen analog. Unterschiedlich ist nur, dass die Betrachtung der Matrix B nun spalten- und nicht mehr zeilenweise geschieht und der Ergebnisvektor nicht mehr Eintrag für Eintrag erstellt wird, sondern die Zwischenergebnisse nach und nach aufsummiert werden.

Multiplikation mit der Steifigkeitsmatrix C

Die effiziente Multiplikation der Steifigkeitsmatrix C mit einem Vektor $\boldsymbol{\alpha}$ gestaltet sich schwieriger. Die Multiplikation der eigentlichen Matrix mit den Skalaren λ und M (Gleichung 2.13) kann für die folgenden Komplexitätsbetrachtungen vernachlässigt werden.

Sei $\phi^1_{\mathbf{l},\mathbf{i}}(\mathbf{x}), \phi^2_{\mathbf{l},\mathbf{i}}(\mathbf{x}), \ldots, \phi^N_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ eine geeignete Auffädelung der Basisfunktionen. Seien α_j die Koeffizienten zur Basis $\{\phi^j_{\mathbf{l},\mathbf{i}}(\mathbf{x})\}_{1 \le j \le N}$. Sei $a(\,.\,,.\,)$ eine Bilinearform.

Dann ist

$$\begin{aligned} C \cdot \boldsymbol{\alpha} &= \begin{pmatrix} a(\phi^1,\phi^1) & a(\phi^1,\phi^2) & a(\phi^1,\phi^3) & \cdots & a(\phi^1,\phi^N) \\ a(\phi^2,\phi^1) & a(\phi^2,\phi^2) & a(\phi^2,\phi^3) & \cdots & a(\phi^2,\phi^N) \\ a(\phi^3,\phi^1) & a(\phi^3,\phi^2) & a(\phi^3,\phi^3) & \cdots & a(\phi^3,\phi^N) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a(\phi^N,\phi^1) & a(\phi^N,\phi^2) & a(\phi^N,\phi^3) & \cdots & a(\phi^N,\phi^N) \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_N \end{pmatrix} \\ &= \begin{pmatrix} a(\phi^1, \sum_{k=1}^N \alpha_k \phi^k) \\ a(\phi^2, \sum_{k=1}^N \alpha_k \phi^k) \\ a(\phi^3, \sum_{k=1}^N \alpha_k \phi^k) \\ \vdots \\ a(\phi^N, \sum_{k=1}^N \alpha_k \phi^k) \end{pmatrix} =: \boldsymbol{\gamma} \end{aligned}$$

und entspricht der Abbildung

$$(\alpha_k)_k \mapsto \left(a(\phi^j, \sum_{k=1}^{N} \alpha_k \phi^k) \right)_j =: (\gamma_j)_j \,. \tag{5.2}$$

Jedes der γ_j ist das Skalarprodukt der Ansatzfunktion ϕ^j mit der Gesamtfunktion

$$f(\mathbf{x}) := \sum_{k=1}^{N} \alpha_k \phi^k$$

zu den Koeffizienten α_k.

Die Basisfunktionen im d-Dimensionalen sind das Tensorprodukt eindimensionaler Hutfunktionen (Gleichung 3.1). Für die Bilinearform $a(\,.\,,.\,)$ gelte für die nächsten Betrachtungen entsprechend

$$a\left(\phi^j_{\mathbf{l},\mathbf{i}}(\mathbf{x}), \phi^k_{\mathbf{l},\mathbf{i}}(\mathbf{x})\right) := \prod_{m=1}^{d} a_m \left(\phi^j_{l_m,i_m}(x_m), \phi^k_{l_m,i_m}(x_m)\right) . \tag{5.3}$$

Der Index m von a deute die Koordinatenrichtung an, in der a angewendet wird.

Die Anwendung der vollständigen Steifigkeitsmatrix C (Abbildung 5.2) kann unterteilt werden in die sukzessive Anwendung von Steifigkeitsmatrizen zu eindimensionalen Basen auf Teile des Vektors $\boldsymbol{\alpha}$. Dies wird zuerst anhand eines einfachen 2×2-Gitters anschaulich gezeigt und anschließend allgemein dargestellt werden.

Ein fiktives 2×2-Gitter in zwei Dimensionen ist in Abbildung 5.2 dargestellt. Die Funktionen $\phi_{(i_1,i_2)}(\mathbf{x})$ sind das Tensorprodukt der beiden eindimensionalen Funktionen $\phi_{i_1}(x_1)$ und $\phi_{i_2}(x_2)$, auf die Angabe des Levels wird verzichtet. Die Anordnung der Basisfunktionen sei lexikographisch nach dem Index $\mathbf{i}$.

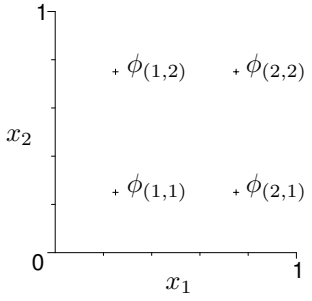


Abbildung 5.2: Ein einfaches 2×2-Gitter.

Es ist

$$C\cdot\boldsymbol{\alpha} = \begin{pmatrix} a(\phi_{(1,1)},\phi_{(1,1)}) & a(\phi_{(1,1)},\phi_{(1,2)}) & a(\phi_{(1,1)},\phi_{(2,1)}) & a(\phi_{(1,1)},\phi_{(2,2)}) \\ a(\phi_{(1,2)},\phi_{(1,1)}) & a(\phi_{(1,2)},\phi_{(1,2)}) & a(\phi_{(1,2)},\phi_{(2,1)}) & a(\phi_{(1,2)},\phi_{(2,2)}) \\ a(\phi_{(2,1)},\phi_{(1,1)}) & a(\phi_{(2,1)},\phi_{(1,2)}) & a(\phi_{(2,1)},\phi_{(2,1)}) & a(\phi_{(2,1)},\phi_{(2,2)}) \\ a(\phi_{(2,2)},\phi_{(1,1)}) & a(\phi_{(2,2)},\phi_{(1,2)}) & a(\phi_{(2,2)},\phi_{(2,1)}) & a(\phi_{(2,2)},\phi_{(2,2)}) \end{pmatrix} \cdot \begin{pmatrix} \alpha_{(1,1)} \\ \alpha_{(1,2)} \\ \alpha_{(2,1)} \\ \alpha_{(2,2)} \end{pmatrix}.$$

Unter Verwendung von (5.3) ergibt dies

$$\begin{aligned} C\cdot\boldsymbol{\alpha} \quad =: \quad & \begin{pmatrix} \gamma_{(1,1)} \\ \gamma_{(1,2)} \\ \gamma_{(2,1)} \\ \gamma_{(2,2)} \end{pmatrix} \\ = \quad & \begin{pmatrix} a_1(\phi_1,\phi_1)\cdot(a_2(\phi_1,\phi_1)\alpha_{(1,1)} + a_2(\phi_1,\phi_2)\alpha_{(1,2)}) \\ \qquad +a_1(\phi_1,\phi_2)\cdot(a_2(\phi_1,\phi_1)\alpha_{(2,1)} + a_2(\phi_1,\phi_2)\alpha_{(2,2)} \\ a_1(\phi_1,\phi_1)\cdot(a_2(\phi_2,\phi_1)\alpha_{(1,1)} + a_2(\phi_2,\phi_2)\alpha_{(1,2)}) \\ \qquad +a_1(\phi_1,\phi_2)\cdot(a_2(\phi_2,\phi_1)\alpha_{(2,1)} + a_2(\phi_2,\phi_2)\alpha_{(2,2)} \\ a_1(\phi_2,\phi_1)\cdot(a_2(\phi_1,\phi_1)\alpha_{(1,1)} + a_2(\phi_1,\phi_2)\alpha_{(1,2)}) \\ \qquad +a_1(\phi_2,\phi_2)\cdot(a_2(\phi_1,\phi_1)\alpha_{(2,1)} + a_2(\phi_1,\phi_2)\alpha_{(2,2)} \\ a_1(\phi_2,\phi_1)\cdot(a_2(\phi_2,\phi_1)\alpha_{(1,1)} + a_2(\phi_2,\phi_2)\alpha_{(1,2)}) \\ \qquad +a_1(\phi_2,\phi_2)\cdot(a_2(\phi_2,\phi_1)\alpha_{(2,1)} + a_2(\phi_2,\phi_2)\alpha_{(2,2)} \end{pmatrix} \end{aligned}$$

oder, anders betrachtet,

$$\begin{pmatrix} \gamma_{(1,1)} \\ \gamma_{(2,1)} \end{pmatrix} = \begin{pmatrix} a_1(\phi_1,\phi_1) & a_1(\phi_1,\phi_2) \\ a_1(\phi_2,\phi_1) & a_1(\phi_2,\phi_2) \end{pmatrix} \begin{pmatrix} \beta_{(1,1)} \\ \beta_{(2,1)} \end{pmatrix}$$
$$\begin{pmatrix} \gamma_{(1,2)} \\ \gamma_{(2,2)} \end{pmatrix} = \begin{pmatrix} a_1(\phi_1,\phi_1) & a_1(\phi_1,\phi_2) \\ a_1(\phi_2,\phi_1) & a_1(\phi_2,\phi_2) \end{pmatrix} \begin{pmatrix} \beta_{(1,2)} \\ \beta_{(2,2)} \end{pmatrix}$$

mit

$$\begin{pmatrix} \beta_{(1,1)} \\ \beta_{(1,2)} \end{pmatrix} = \begin{pmatrix} a_2(\phi_1,\phi_1) & a_2(\phi_1,\phi_2) \\ a_2(\phi_2,\phi_1) & a_2(\phi_2,\phi_2) \end{pmatrix} \begin{pmatrix} \alpha_{(1,1)} \\ \alpha_{(1,2)} \end{pmatrix}$$
$$\begin{pmatrix} \beta_{(2,1)} \\ \beta_{(2,2)} \end{pmatrix} = \begin{pmatrix} a_2(\phi_1,\phi_1) & a_2(\phi_1,\phi_2) \\ a_2(\phi_2,\phi_1) & a_2(\phi_2,\phi_2) \end{pmatrix} \begin{pmatrix} \alpha_{(2,1)} \\ \alpha_{(2,2)} \end{pmatrix}.$$

Dies entspricht der Anwendung der eindimensionalen Steifigkeitsmatrix zuerst für alle Gitterpunkte mit dem selben x_1-Wert in x_2-Richtung und anschließend auf den Ergebnisvektor für alle Punkte mit dem selben Wert von x_2 in x_1-Richtung, vergleiche Abbildung 5.3.

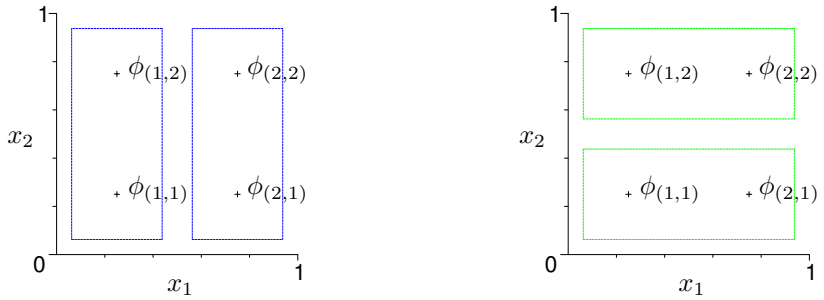


Abbildung 5.3: Anwendung der eindimensionalen Steifigkeitsmatrizen in x_2- und x_1-Richtung auf ein 2×2-Gitter.

Zuerst wird der Einfluss aller Basisfunktionen auf alle anderen Basisfunktionen in x_2-Richtung berechnet, dann die gesammelten Werte in x_1-Richtung propagiert und um die Einflüsse in dieser Richtung ergänzt.

Allgemein formuliert bedeutet dies bei einem regelmäßigen $n \times n$-Gitter im Zweidimensionalen mit den Basisfunktionen $\{\phi_{(i_1,i_2)}\}_{1 \leq i_1,i_2 \leq n}$: Bei der Anwendung der eindimensionalen Steifigkeitsmatrix auf jede Spalte i_1 werden für jedes i_1 aus den Koeffizienten $\alpha_{(i_1,i_2)}$ die Zwischenwerte $\beta_{(i_1,i_2)}$ mit

$$\beta_{(i_1,i_2)} := \sum_{1 \leq j_2 \leq n} \alpha_{(i_1,j_2)} a_2(\phi_{i_2}, \phi_{j_2}), \quad 1 \leq i_2 \leq n. \tag{5.4}$$

Im zweiten Schritt werden bei der zeilenweisen Anwendung für jede Zeile i_2 aus den neuen Koeffizienten $\beta_{(i_1,i_2)}$ die Ergebniswerte $\gamma_{(i_1,i_2)}$ mit

$$\gamma_{(i_1,i_2)} := \sum_{1 \leq j_1 \leq n} \beta_{(j_1,i_2)} a_1(\phi_{i_1}, \phi_{j_1}), \quad 1 \leq i_1 \leq n. \tag{5.5}$$

Gleichung 5.4 eingesetzt ergibt

$$\begin{aligned}
\gamma_{(i_1,i_2)} &= \sum_{1 \leq j_1 \leq n} \left(\sum_{1 \leq j_2 \leq n} \alpha_{(j_1,j_2)} a_2(\phi_{i_2}, \phi_{j_2}) \right) a_1(\phi_{i_1}, \phi_{j_1}) \\
&= \sum_{1 \leq j_1, j_2 \leq n} \alpha_{(j_1,j_2)} a_1(\phi_{i_1}, \phi_{j_1}) a_2(\phi_{i_2}, \phi_{j_2}) \\
&= \sum_{1 \leq i_1, i_2 \leq n} \alpha_{(j_1,j_2)} a(\phi_{(i_1,i_2)}, \phi_{(j_1,j_2)}) \\
&= a(\phi_{(i_1,i_2)}, \sum_{1 \leq i_1, i_2 \leq n} \alpha_{(j_1,j_2)} \phi_{(j_1,j_2)}),
\end{aligned}$$

was wieder der Anwendung der gesamten Steifigkeitsmatrix C entspricht (Gleichung 5.2).

Bei einem d-dimensionalen Merkmalsraum kann die Multiplikation mit der Steifigkeitsmatrix C analog in mehrfache Anwendung einer eindimensionalen Steifigkeitsmatrix unterteilt werden. In jeder Koordinatenrichtung muss für jede Gitterstelle der eindimensionale Fall sukzessive angewendet werden, wodurch in d Schritten aus dem Vektor α der Ergebnisvektor γ wird. Die Reihenfolge, in der die Koordinatenrichtungen x_i berücksichtigt werden, ist durch die Kommutativität der Teilprodukte der Bilinearform beliebig.

Im Vergleich mit einem Verfahren, das die Gesamtsteifigkeitsmatrix C aufstellen würde, würde sich die Anzahl der Operationen durch die Unterteilung in Teilprobleme von $\mathcal{O}(n^{2d})$ auf $\mathcal{O}(d \cdot n^{d+1})$ reduzieren. Es muss bei n Gitterpunkten pro Richtung nicht mehr eine $n^d \times n^d$-Steifigkeitsmatrix aufgestellt werden, sondern nur noch eine $n \times n$-Steifigkeitsmatrix, die mehrfach verwendet wird. Im Hinblick auf Anzahl der Operationen, Speicherverbrauch und Caching-Verhalten kann bereits hiermit eine Verbesserung erzielt werden.

Betrachte nun die Bilinearform

$$a(\phi^j, \phi^k) := (\nabla\phi^j, \nabla\phi^k)_{L_2} = \int_\Omega \nabla\phi^j_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \nabla\phi^k_{\mathbf{l},\mathbf{i}}(\mathbf{x}) d\mathbf{x}. \tag{5.6}$$

Diese Bilinearform kann nicht direkt als Produkt von Bilinearformen für jede Koordinatenrichtung geschrieben werden. Hier gilt:

$$\begin{aligned}
a(\phi^j, \phi^k) &= \int \nabla\phi^j_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \nabla\phi^k_{\mathbf{l},\mathbf{i}}(\mathbf{x}) d\mathbf{x} \\
&= \int \frac{\partial}{\partial x_1}\phi^j_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \frac{\partial}{\partial x_1}\phi^k_{\mathbf{l},\mathbf{i}}(\mathbf{x}) + \cdots + \frac{\partial}{\partial x_d}\phi^j_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \frac{\partial}{\partial x_d}\phi^k_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \; d\mathbf{x}
\end{aligned}$$

$$
\begin{aligned}
a(\phi^j,\phi^k) &= \int \Bigg(\frac{\partial}{\partial x_1}\phi^j_{l_1,i_1}(x_1)\frac{\partial}{\partial x_1}\phi^k_{l_1,i_1}(x_1)\cdot \phi^j_{l_2,i_2}(x_2)\phi^k_{l_2,i_2}(x_2)\cdot \ldots \cdot \phi^j_{l_d,i_d}(x_d)\phi^k_{l_d,i_d}(x_d) \\
&\quad + \cdots + \\
&\quad \phi^j_{l_1,i_1}(x_1)\phi^k_{l_1,i_1}(x_1)\cdot \ldots \cdot \phi^j_{l_{d-1},i_{d-1}}(x_{d-1})\phi^k_{l_{d-1},i_{d-1}}(x_{d-1}) \\
&\qquad\qquad \cdot \frac{\partial}{\partial x_d}\phi^j_{l_d,i_d}(x_d)\frac{\partial}{\partial x_d}\phi^k_{l_d,i_d}(x_d)\Bigg)\, d\mathbf{x} \\
&= \int \frac{\partial}{\partial x_1}\phi^j_{l_1,i_1}(x_1)\frac{\partial}{\partial x_1}\phi^k_{l_1,i_1}(x_1)\, dx_1 \\
&\qquad \cdot \int \phi^j_{l_2,i_2}(x_2)\phi^k_{l_2,i_2}(x_2)\, dx_2 \cdot \ldots \cdot \int \phi^j_{l_d,i_d}(x_d)\phi^k_{l_d,i_d}(x_d)\, dx_d \\
&\quad + \cdots + \\
&\quad \int \phi^j_{l_1,i_1}(x_1)\phi^k_{l_1,i_1}(x_1)\, dx_1 \cdot \ldots \cdot \int \phi^j_{l_{d-1},i_{d-1}}(x_{d-1})\phi^k_{l_{d-1},i_{d-1}}(x_{d-1})\, dx_{d-1} \\
&\qquad \cdot \int \frac{\partial}{\partial x_d}\phi^j_{l_d,i_d}(x_d)\frac{\partial}{\partial x_d}\phi^k_{l_d,i_d}(x_d)\, dx_d \\
&= a_1(\phi^j,\phi^k)\cdot \tilde{a}_2(\phi^j,\phi^k)\cdot \ldots \cdot \tilde{a}_d(\phi^j,\phi^k) \\
&\quad + \cdots + \\
&\quad \tilde{a}_1(\phi^j,\phi^k)\cdot \ldots \cdot \tilde{a}_{d-1}(\phi^j,\phi^k)\cdot a_d(\phi^j,\phi^k) \\
&= \sum_{s=1}^{d} a_s(\phi^j,\phi^k) \prod_{\substack{1\le t\le d,\\ t\neq s}} \tilde{a}_t(\phi^j,\phi^k),
\end{aligned}
$$

mit

$$\tilde{a}_t(\phi^j,\phi^k) := (\phi^j_{l_t,i_t},\phi^k_{l_t,i_t})_{L_2}.$$

Für obigen Algorithmus bedeutet dies, dass er für jeden Summanden getrennt durchgeführt werden muss, also insgesamt d mal. Die Ergebnisse werden aufaddiert. Bei jeder Durchführung wird in einer der Koordinatenrichtungen eine andere Steifigkeitsmatrix zur eindimensionalen Basis verwendet als in den anderen Richtungen.

Die bisherigen Betrachtungen beschränken sich auf volle Gitter und sind nicht abhängig von einer speziellen Basis. Es wird nur die Tensorproduktkonstruktion höherdimensionaler Basisfunktionen benötigt. Nun sollen dünne Gitter betrachtet werden. Zur Vereinfachung seien die Randwerte null: Verwendet wird die hierarchische Basis ohne Berücksichtigung des Randes aus Kapitel 4.2. Die vorgestellten Verfahren lassen sich auf beliebige Randwerte erweitern.

Abbildung 5.4 zeigt die Anwendung auf das dünne Gitter zum Dünngitterraum $V_2^{(1)}$. Entsprechend zu dem 2×2-Gitter in Abbildung 5.3 sollen zuerst für jede Spalte x_1 die Steifigkeitsmatrizen zu den eindimensionalen Basen angewendet werden, also für alle auftretenden (l_1,i_1)-Kombinationen, nämlich $(2,1)$, $(1,1)$ und $(2,3)$. Hier werden jedoch für die einzelnen Spalten unterschiedliche Steifigkeitsmatrizen verwendet. Für die erste und dritte Spalte ist es die 1×1-Matrix zur hierarchischen Basis mit Level 1, für die mittlere Spalte die 3×3-Matrix zum Level 2. Bei der zeilenweisen Anwendung geschieht dies analog.

Bei dünnen Gittern funktioniert jedoch das Propagieren der gesammelten Informationen zwischen den einzelnen Schritten nicht immer. Bei der Anordnung der beiden Schritte in Abbildung 5.4 wird der Beitrag der Basisfunktion $\phi_{(\frac{2}{1})(\frac{3}{1})}$ zur Gesamtfunktion nicht an die Basisfunktion $\phi_{(\frac{1}{2})(\frac{1}{3})}$ weitergereicht. Bei einem vollbesetzten Gitter würde dieser im ersten Schritt im Koeffizienten der Basisfunktion $\phi_{(\frac{2}{2})(\frac{3}{3})}$ zwischengespeichert und im zweiten Schritt in der obersten Zeile weitergereicht. Der Gitterpunkt dieser Basisfunktion fehlt im dünnen Gitter.

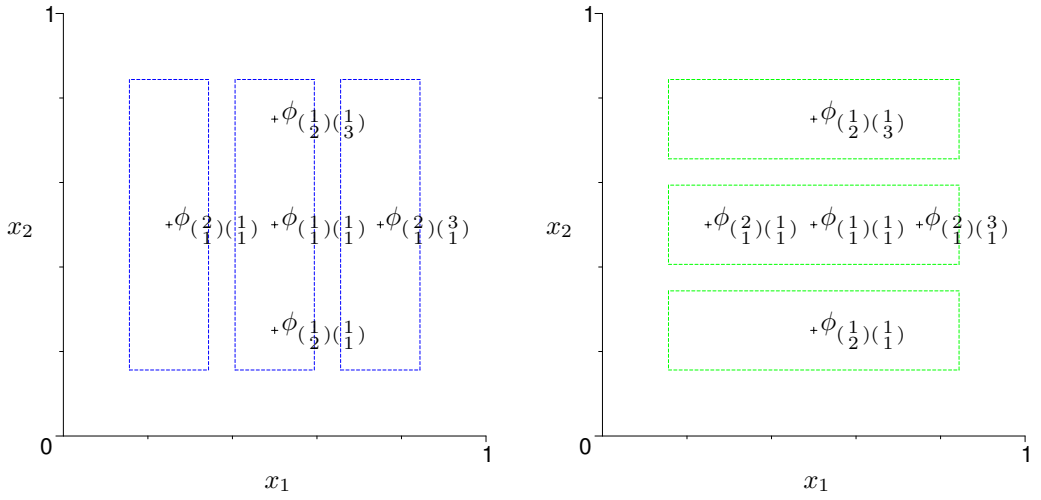


Abbildung 5.4: Anwendung der eindimensionalen Steifigkeitsmatrizen in x_2- und x_1-Richtung auf das dünne Gitter zum Raum $V_2^{(1)}$.

Die Rückrichtung, die Weitergabe der Information der Basisfunktion $\phi_{(\frac{1}{2})(\frac{1}{3})}$ an die Funktion $\phi_{(\frac{2}{1})(\frac{3}{1})}$, bereitet keine Probleme, da die Information im Gitterpunkt zu $\phi_{(\frac{1}{1})(\frac{1}{1})}$ zwischengespeichert wird und daher nicht verloren geht.

Abhilfe für das Problem der Informationspropagierung zwischen den Dimensionen bereitet das sogenannte *up und down*-Verfahren, das auf der hierarchischen Struktur der Basisfunktionen beruht.

5.3.2 Das up und down-Verfahren im Eindimensionalen

Betrachte nun den Informationsfluss beim ersten Schritt in der mittleren Spalte in Abbildung 5.4. Durch den hierarchischen Aufbau der Basis gibt es keinen Informationsfluss zwischen den Basisfunktionen $\phi_{(\frac{1}{2})(\frac{1}{1})}$ und $\phi_{(\frac{1}{2})(\frac{1}{3})}$: Die Träger der beiden Basisfunktionen in x_2-Richtung sind disjunkt, d. h. die Bilinearformen $a_2(\phi_{(\frac{1}{2})(\frac{1}{1})}, \phi_{(\frac{1}{2})(\frac{1}{3})})$ und $\tilde{a}_2(\phi_{(\frac{1}{2})(\frac{1}{1})}, \phi_{(\frac{1}{2})(\frac{1}{3})})$ liefern den Wert null.

Information wird zwischen den Basisfunktionen $\phi_{(\frac{1}{2})(\frac{1}{1})}$ und $\phi_{(\frac{1}{1})(\frac{1}{1})}$ ausgetauscht, sowie zwischen $\phi_{(\frac{1}{2})(\frac{1}{3})}$ und $\phi_{(\frac{1}{1})(\frac{1}{1})}$. Stellt man sich die Komponenten der Basisfunktionen in x_2-Richtung als Binärbaum vor (vergleiche Kapitel 3.2), so fließt nur Information von allen Knoten zu allen hierarchisch höher und tiefer liegenden Knoten. Es existiert kein Austausch zwischen Knoten benachbarter Teilbäume, sondern nur entlang der Wege von den Blättern zur Wurzel und umgekehrt.

Seien die eindimensionalen Hutfunktionen $\phi_{l,i}$ bis zum Level n und die zugehörigen Unbekannten $\alpha_{l,i}$ nach Größe geordnet, d. h. kleine Level zuerst. Dann kann die Steifigkeitsmatrix C zerlegt werden in

$$C := C^{\text{up}} + C^{\text{down}}, \tag{5.7}$$

wobei C^{up} die obere Dreiecksmatrix und C^{down} die untere Dreiecksmatrix inklusive der Diagonalen ist.

Die Anwendung des Operators C^{up} auf den Vektor $\boldsymbol{\alpha}$ beschreibt den Informationsfluss von hierarchisch niedrigeren zu höheren Basisfunktionen:

$$\left(\beta_{l,i}^{\text{up}}\right) := \left(a(\phi_{l,i}, \sum_{l'>l} \alpha_{l',i'}\phi_{l',i'})\right). \tag{5.8}$$

Entsprechend beschreibt die Anwendung der Matrix C^{down} den Einfluss von höheren zu niedrigeren Basisfunktionen (Basisfunktionen auf dem selben Level beeinflussen nur sich selbst):

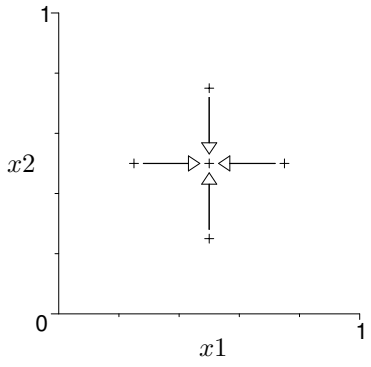
$$\left(\beta_{l,i}^{\text{down}}\right) := \left(a(\phi_{l,i}, \sum_{l'\leq l} \alpha_{l',i'}\phi_{l',i'})\right). \tag{5.9}$$

Zusammen ergibt sich wieder $\boldsymbol{\beta}$:

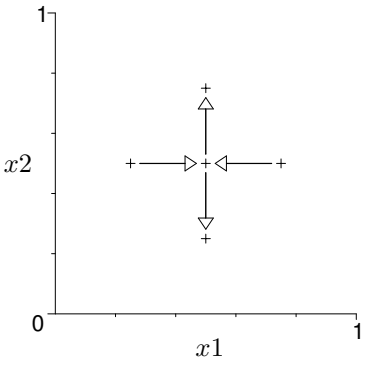
$$\boldsymbol{\beta} = C \cdot \boldsymbol{\alpha} = C^{\text{up}} \cdot \boldsymbol{\alpha} + C^{\text{down}} \cdot \boldsymbol{\alpha} = \boldsymbol{\beta}^{\text{up}} + \boldsymbol{\beta}^{\text{down}}. \tag{5.10}$$

5.3.3 Das up und down-Verfahren im d-Dimensionalen

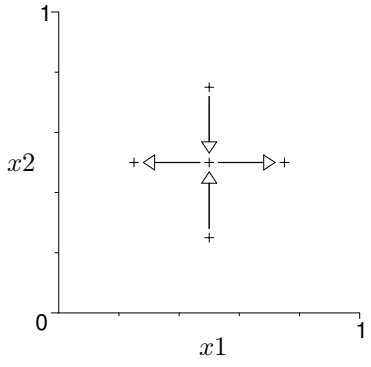
Im d-dimensionalen Merkmalsraum wird in jeder Richtung entweder up oder down ausgeführt. Da beide Komponenten des 1-D Operators C benötigt werden, muss dies im Gesamtalgorithmus für alle möglichen Kombinationen von up und down durchgeführt werden. Abbildung 5.5 zeigt den Informationsfluss für zwei Dimensionen.



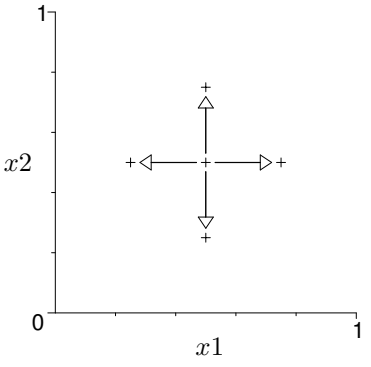
up in x_1-, up in x_2-Richtung



up in x_1-, down in x_2-Richtung



up in x_2-, down in x_1-Richtung(!)



down in x_1-, down in x_2-Richtung

Abbildung 5.5: Informationsfluss für up und down zum Dünngitterraum $V_2^{(1)}$ in 2D.

Sei up_j ein Bezeichner für die Anwendung des Operators C^{up} in Koordinatenrichtung x_j, $down_j$ für die Anwendung des Operators C^{down}. Sei $up_j\ down_k$ die Hintereinanderausführung von *up* in Richtung x_j und *down* in Richtung x_k.

In d Dimensionen müssen 2^d Kombinationen von *up* und *down*-Durchführungen berücksichtigt werden. Bei zwei Dimensionen sind es

$$\begin{aligned} up_1\ up_2, \\ up_1\ down_2, \\ down_1\ up_2 \qquad &\text{und} \\ down_1\ down_2. \end{aligned}$$

Dies entspricht einer zuerst zeilenweisen, dann spaltenweisen Anwendung der eindimensionalen Steifigkeitsmatrizen. Analog zum Problem der Informationsweitergabe von Basisfunktion $\phi_{(\frac{2}{1})(\frac{3}{1})}$ zu Basisfunktion $\phi_{(\frac{1}{2})(\frac{1}{3})}$ in Abbildung 5.3 funktioniert diese nun nicht bei der Rückrichtung, der Durchführung von $down_1\ up_2$. Abbildung 5.5 zeigt, dass die Weitergabe möglich ist, und zwar über die gemeinsame Wurzel in Baumdarstellung: Zuerst muss die benötigte Information von $\phi_{(\frac{1}{2})(\frac{1}{3})}$ in x_2-Richtung an die Wurzel, den Gitterpunkt zu $\phi_{(\frac{1}{1})(\frac{1}{1})}$, weitergegeben und dort zwischengespeichert werden, und kann dann erst in x_1-Richtung den Funktionsbaum entlang an $\phi_{(\frac{2}{1})(\frac{3}{1})}$ hinuntergereicht werden: Es muss $up_2\ down_1$ anstelle von $down_1\ up_2$ durchgeführt werden.

Es gilt im allgemeinen Fall: Alle Wörter

$$\{up|down\}_1\ \{up|down\}_2\ \cdots \{up|down\}_d,$$

in denen ein *down*-Schritt vor einem *up*-Schritt vorkommt, müssen durch eine Permutation ersetzt werden, in der dies nicht der Fall ist.

Bei $\mathcal{O}(2^d)$ Durchführungen von *up*- bzw. *down*-Schritten ist eine effiziente Realisierung wichtig. Jede von diesen kann in $\mathcal{O}(N)$ Operationen realisiert werden.

5.3.4 Effiziente Realisierung von *up* und *down*

Betrachte nun die Anwendung des *up*- bzw. *down*-Algorithmus in einer Koordinatenrichtung. Seien die Basisfunktionen $\{\phi_{l,i}\}$ die Komponenten der betroffenen Basisfunktionen in dieser Richtung mit den Koeffizienten $\alpha_{l,i}$. Die Anwendung des *up*-Operators C^{up} ist für jede Basisfunktion $\phi_{l,i}$ das Bilden des Skalarprodukts von dieser mit der gewichteten Summe aller Basisfunktionen mit größerem Level (Gleichung 5.8), entsprechend die Anwendung des *down*-Operators das Skalarprodukt mit der gewichteten Summe aller Basisfunktionen mit selbem oder kleinerem Level (Gleichung 5.9).

Der 1-D-Operator C enthält für die Bilinearform $a(\,.\,,.\,)$ nur Einträge ungleich null auf der Diagonalen. Es ist

$$\begin{aligned} a(\phi_{l,i}, \phi_{l',i'}) &= \int \frac{\partial}{\partial x}\phi_{l,i}(x) \cdot \frac{\partial}{\partial x}\phi_{l',i'}(x)\ dx \\ &= \begin{cases} 2^{l+1} & \text{für } l = l',\ i = i', \\ 0 & \text{sonst.} \end{cases} \end{aligned} \tag{5.11}$$

Die Durchführung eines *down*-Schrittes beschränkt sich somit auf das Multiplizieren aller $\alpha_{l,i}$ mit 2^{l+1}. Ein *up*-Schritt ergibt den Nullvektor.

Die Anwendung des 1-D-Operators C für die Bilinearform

$$\tilde{a}(\phi_{l,i}, \phi_{l',i'}) = \int \phi_{l,i}(x) \cdot \phi_{l',i'}(x)\ dx \tag{5.12}$$

benötigt eine nähere Untersuchung.

Abbildung 5.6 zeigt dies für die Basisfunktion $\phi_{3,3}$. Das Produkt mit der Gesamtfunktion $f(x)$ wird unterteilt in das Produkt mit Basisfunktionen kleineren Levels für *down* (blau) und in das Produkt mit Basisfunktionen größeren Levels für *up* (grün). Das Produkt mit anderen Basisfunktionen auf dem selben Level ergibt null, da die Träger disjunkt sind. Die einzige Funktion mit gleichem Level, die für *down* berücksichtigt werden muss, ist die Hutfunktion $\phi_{3,3}$ selbst (schwarz).

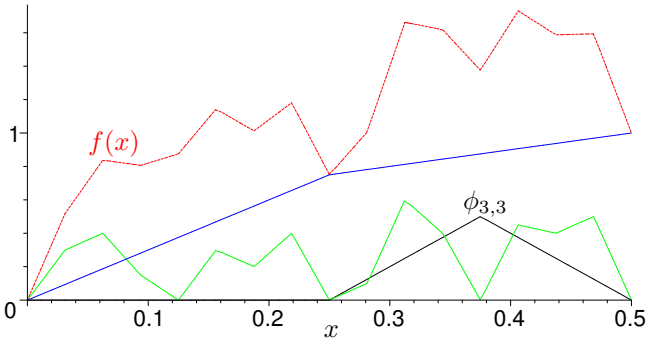


Abbildung 5.6: Die Gesamtfunktion $f(x)$ (rot gestrichelt), der Anteil der Basisfunktionen mit $l < 3$ (blau), $l > 3$ (grün) und die gewichtete Ansatzfunktion $\phi_{3,3}(x)$.

Es müssen nur Basisfunktionen, die auf dem Träger der Ansatzfunktion nicht konstant null sind, berücksichtigt werden. Dies sind in der Binärbaumdarstellung für *down* alle Basisfunktionen auf dem Weg von der Wurzel zu $\phi_{3,3}$ und für *up* alle Funktionen, die im Baum unterhalb von $\phi_{3,3}$ liegen. Abbildung 5.7 zeigt den Baum der Basisfunktionen bis Level $l = 4$.

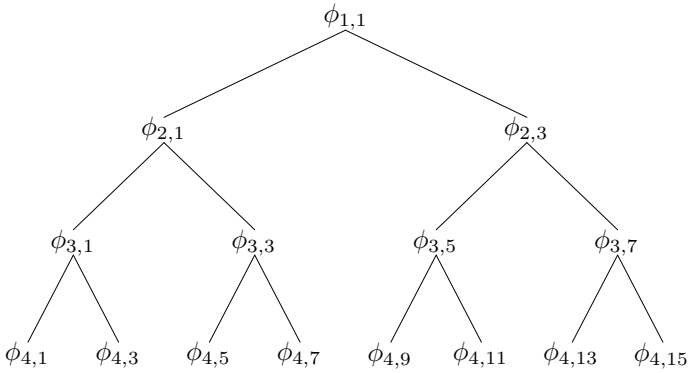


Abbildung 5.7: Binärbaum der Basisfunktionen $\phi_{l,i}$ bis Level $l = 4$.

5.3.5 Der *down*-Algorithmus

Die Summe aller gewichteten, hierarchisch höheren Basisfunktionen (alle mit kleinerem Level) ergibt auf dem Träger der Hutfunktion $\phi_{l,i}$ eine lineare Funktion mit den Randwerten f_l und f_r an der Stellen $x_l = (i-1)2^{-l}$ und $x_r = (i+1)2^{-l}$, siehe Abbildung 5.8.

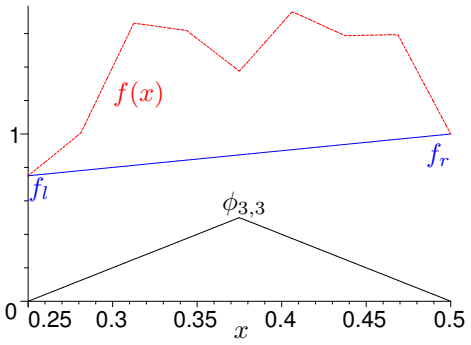


Abbildung 5.8: Die Gesamtfunktion $f(x)$ (rot gestrichelt), der Anteil der Basisfunktionen mit $l < 3$ (blau) und die gewichtete Ansatzfunktion $\phi_{3,3}(x)$.

Betrachte das L_2-Skalarprodukt der Hutfunktion $\phi_{1,1}$ mit einer linearen Funktion. Jede Basisfunktion $\phi_{l,i}$ kann mittels Translation von x_l zum Nullpunkt und anschließender Streckung in x-Richtung um den Faktor $1/(2h) = 2^{l-1}$ auf diese Referenzfunktion abgebildet werden. Entsprechendes gilt für jede lineare Funktion, die durch $f(x_l) := f_l$ und $f(x_r) := f_r$ definiert ist. Es gilt dann mit $x_l = 0$ und $x_r = 1$:

$$\tilde{a}\big(\phi_{1,1}(x), f_l + (f_r - f_l)x\big) = \int_{x_l}^{x_r} \phi_{1,1}(x) \cdot (f_l + (f_r - f_l)x)\; dx \tag{5.13}$$

$$= \tilde{a}\big(\phi_{1,1}(x), (f_l + f_r)/2\big) = \int_{x_l}^{x_r} \phi_{1,1}(x) \cdot (f_l + f_r)/2\; dx \tag{5.14}$$

$$= \tilde{a}\big(1/2, f_l + (f_r - f_l)x\big) = \int_{x_l}^{x_r} 1/2 \cdot (f_l + (f_r - f_l)x)\; dx \tag{5.15}$$

$$= \tilde{a}\big(1/2, (f_l + f_r)/2\big) = \int_{x_l}^{x_r} 1/2 \cdot (f_l + f_r)/2\; dx \tag{5.16}$$

$$= (f_l + f_r)/4. \tag{5.17}$$

Die Rücktransformation entspricht der Multiplikation des Intergralwertes mit $2h = 2^{-l+1}$. Es kann also wahlweise die Hutfunktion durch die konstante Funktion $f(x) := 1/2$ oder die lineare Funktion durch die konstante Funktion $f(x) := (f_l + f_r)/2$ ersetzt werden, siehe auch Abbildung 5.9.

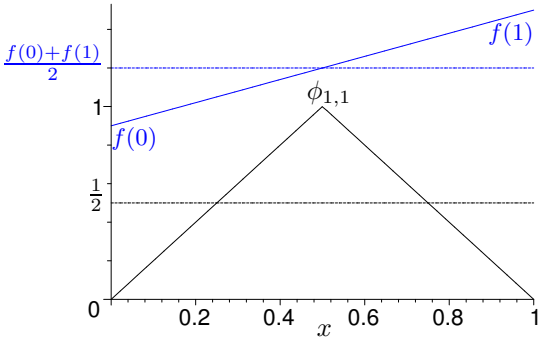


Abbildung 5.9: Alternativen für das L_2-Skalarprodukt.

Sind beim *down*-Schritt bei der Betrachtung der Ansatzfunktion $\phi_{l,i}$ die Werte f_l und f_r bekannt, so berechnet sich der neue Koeffizientenwert zu

$$\beta_{l,i} := \frac{1}{4}(f_l + f_r) \cdot 2h.$$

Da der *down*-Operator C^{down} die Diagonalelemente von C enthält, entsteht ein zusätzlicher Term

$$\tilde{a}(\phi_{l,i}, \phi_{l,i}) = \frac{1}{3} \cdot 2h \cdot \alpha_{l,i}$$

und folglich insgesamt für den Koeffizienten der Ansatzfunktion $\phi_{l,i}$

$$\beta_{l,i} := \frac{1}{4}(f_l + f_r) \cdot 2h + \frac{1}{3} \cdot 2h \cdot \alpha_{l,i}. \tag{5.18}$$

Die Berechnung der Randwerte f_l und f_r kann ähnlich zu der Berechnung der Funktionswerte an den Gitterstellen aus den hierarchischen Überschüssen durchgeführt werden: Es genügt eine Traversierung des Baumes der Basisfunktionen. Hier interessieren jedoch die Werte an den Randpunkten der Hutfunktionen und nicht an den Gitterpunkten. Die Randwerte der Wurzelfunktion $\phi_{1,1}$ sind null. Bis zum Erreichen des maximalen Levels wird rekursiv im Baum abgestiegen. Beim Abstieg in den linken Teilbaum wird der rechte Randwert verändert, beim Abstieg in den rechten Teilbaum der linke.

Sei $\phi_{l,i}$ die aktuell betrachtete Funktion mit den Randwerten f_l und f_r. Beim Abstieg zum linken Kind $\phi_{l+1,2i-1}$ bleibt der selbe linke Randwert erhalten, $f'_l := f_l$. Der neue rechte Randwert ergibt sich zu $f'_r := (f_l + f_r)/2 + \alpha_{l,i}$. Entsprechend ist für das rechte Kind $\phi_{l+1,2i+1}$ der rechte Randwert $f'_r := f_r$ und der linke Randwert $f'_l := (f_l + f_r)/2 + \alpha_{l,i}$.

Für die Komplexitätsbetrachtung ist wichtig, dass für jeden Knoten nur eine konstante Zahl von Operationen benötigt wird. Insbesondere muss auf die Koeffizienten $\alpha_{l,i}$ und $\beta_{l,i}$ in konstanter Zeit zugegriffen werden können. Sie gehören jedoch zu den d-dimensionalen Basisfunktionen $\phi_{\mathbf{l},\mathbf{i}}$ und stehen in einem Vektor $\boldsymbol{\alpha}$ bzw. $\boldsymbol{\beta}$ der Länge N in der Reihenfolge, die durch die Auffädelung der Basisfunktionen festgelegt ist. Es darf nicht für jeden Zugriff der entsprechende Vektor durchsucht werden.

Im Paket `DG` wird nach der Erstellung der Liste der Gitterpunkte (Kapitel 5.2) eine Hashfunktion aufgestellt, die zu einer Liste $[l_1, i_1, \ldots, l_d, i_d]$, d. h. zu einem Paar Multiindizes $(\mathbf{l}, \mathbf{i})$, den Index von $\phi_{\mathbf{l},\mathbf{i}}$ in der Auffädelung der Basisfunktionen liefert. Dies benötigt einmalig $\mathcal{O}(N)$ Schritte. Das Finden des Indizes zu gegebenem $\mathbf{l}$ und $\mathbf{i}$ erfolgt anschließend in $\mathcal{O}(1)$.

Listing 5.2 zeigt die *down*-Prozedur, die die rekursiven Baumtraversierungen in Richtung $x_{\texttt{dim}}$ für den Dünngitterraum $V^{(1)}_{\texttt{level}}$ initialisiert. Für den d-dimensionalen Merkmalsraum wird die Liste der Gitterpunkte zum $(d-1)$-dimensionalen dünnen Gitter aufgestellt. Für jede Baumtraversierung in Richtung `dim` sind die Indizes in den anderen Richtungen fix. Die $(d-1)$-dimensionalen Gitterpunkte stellen die Listen der Indizes

$$[l_1, i_1, \ldots, l_{\texttt{dim}-1}, i_{\texttt{dim}-1}, l_{\texttt{dim}+1}, i_{\texttt{dim}+1}, \ldots, l_d, i_d]$$

dar. Jede Indexliste legt die Tiefe des Baumes in $x_{\texttt{dim}}$-Richtung eindeutig fest. Für das dünne Gitter aus Abbildung 5.3 ist beim Schritt $down_2$ die Liste der Indizes für die x_1-Richtung $[[1,1],[2,1],[2,3]]$ mit den Baumtiefen in x_2-Richtung 2, 1 und 1.

Listing 5.3 zeigt die Prozedur `down_rec` zur rekursiven Baumtraversierung für $down_{\texttt{dim}}$. In jedem Knoten $\phi_{l,i}$ wird der neue Koeffizientenwert $\beta_{l,i}$ berechnet und rekursiv in den linken und rechten Teilbaum abgestiegen.

```
down := proc(dim::integer, level::integer, Alpha::Vector, lbf::list,
             indexhash::table)
  # dim       ... Dimension, in der down ausgeführt werden soll
  # level     ... Maximales Level
  # Alpha     ... Koeffizientenvektor
  # indexhash ... Hashfunktion [(Level, Index)*] -> Index der d-dim. Basisfkt.
  #
  ## Rückgabe:
  # Beta      ... Ergebnisvektor
  local d, lbf_dminus1, bf, Beta;

  # Dimension des Merkmalraums ermitteln:
  d := nops(lbf[1])/2;

  # Ergebnisvektor Beta erstellen:
  Beta := Vector(Dimension(Alpha));

  # Liste der Basisfunktionen im (d-1)-dimensionalen Raum erstellen:
  lbf_dminus1 := li_liste_nb(d-1, level);

  # für jede (d-1)-dimensionale Basisfunktion in Dimension dim im Baum absteigen
  for bf in lbf_dminus1 do
    down_rec(0, 0, level-add(bf[2*i-1], i=1..d-1)+d-1, 1, 1,
             bf[1..2*(dim-1)], bf[2*(dim)-1..2*(d-1)], Alpha, Beta, indexhash);
  end do;

  # Rückgabe:
  return Beta;

end proc:
```

Listing 5.2: Prozedur für $down_{\texttt{dim}}$.

```
down_rec := proc(fl, fr, level::integer, akt_level::integer, akt_ind::integer,
                 praefix::list, postfix::list,
                 Alpha::Vector, Beta::evaln(Vector), indexhash::table)
  # fl, fr              ... Funktionswert an der linken (rechten) Intervallgrenze
  # level               ... Maximales Level
  # akt_level, akt_ind ... Level und Index der akt. Basisfunktion (für Rekursion)
  # praefix, postfix    ... [(Level, Basisfunktion)*] der vorigen
  #                         (nachfolgenden) Dimensionen
  # Alpha               ... Koeffizientenvektor
  # Beta                ... Ergebnisvektor
  # indexhash           ... Hashfunktion [(Level, Index)*] -> Index der Basisfkt.

  local li_index;
  # Gesamtindex der aktuellen Basisfunktion bestimmen
  li_index := indexhash[[op(praefix), akt_level, akt_ind, op(postfix)]]:

  Beta[li_index] :=
    ((fr+fl)/4)/2^(akt_level-1)                    # Interpolant * Testfunktion
    + 1/3 * 1/2^(akt_level-1) * Alpha[li_index];    # Diagonale

  # Abbruchbedingung (weiterer Abstieg?):
  if(akt_level < level) then

    # Abstieg links
    down_rec(fl, (fl+fr)/2 + Alpha[li_index], level, akt_level+1, akt_ind*2-1,
             praefix, postfix, Alpha, Beta, indexhash);

    # Abstieg rechts
    down_rec((fl+fr)/2 + Alpha[li_index], fr, level, akt_level+1, akt_ind*2+1,
             praefix, postfix, Alpha, Beta, indexhash);

  end if;
end proc:
```

Listing 5.3: Rekursive Baumtraversierung für $down_{\texttt{dim}}$.

5.3.6 Der *up*-Algorithmus

Für den *up*-Schritt ist die Summe aller gewichteten, hierarchisch niedrigeren Basisfunktionen (alle mit größerem Level) leider nicht linear auf dem Träger der Ansatzfunktion $\phi_{l,i}$. Sie ist vielmehr stückweise linear mit Maschenweite $h_n = 2^{-n}$ für das maximal verwendete Abstiegslevel n. Die Funktionswerte am linken Rand des Trägers (f_l), am rechten Rand (f_r), sowie am Gitterpunkt der Basisfunktion (f_m) sind null, vergleiche Abbildung 5.10.

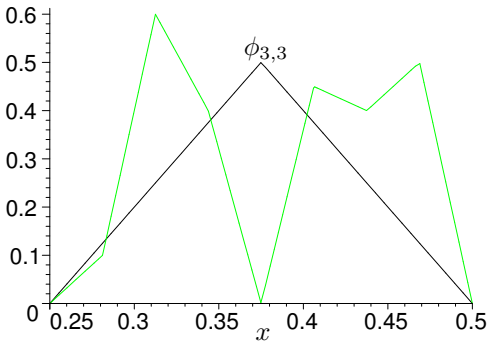


Abbildung 5.10: Der Anteil der Basisfunktionen mit $l > 3$ (grün) und die gewichtete Ansatzfunktion $\phi_{3,3}(x)$.

Die einzelnen Basisfunktionen müssen nun getrennt behandelt werden. Abbildung 5.11 zeigt die selbe stückweise lineare Funktion zerlegt in ihre Teilfunktionen. Entsprechend zu den Überlegungen zu den Gleichungen 5.13 bis 5.17 übernimmt hier die Ansatzfunktion die Rolle der linearen Funktion. Die Beiträge der hierarchisch niedrigeren Funktionen müssen den Funktionsbaum emporgereicht werden.

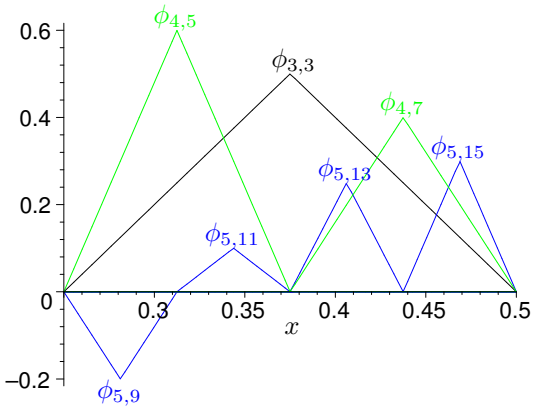


Abbildung 5.11: Die gewichteten Basisfunktionen mit $l > 3$.

Zur Berechnung des L_2-Skalarproduktes der Ansatzfunktion mit einer der Hutfunktionen $\phi_{l',i'}$ wird zum einen der halbe Wert des Koeffizienten $\alpha_{l',i'}$ multipliziert mit der Breite des Trägers, $2^{-l'+1}$, benötigt. Dies entspricht dem Beitrag der Hutfunktion (Abbildung 5.9). Zum anderen wird der Funktionswert der Ansatzfunktion am Gitterpunkt der Hutfunktion $\phi_{l',i'}$ benötigt, was dem Wert $(f_l + f_r)/2$ entspricht. In Abbildung 5.11 sind dies die Werte

$$\frac{1}{4} \cdot \alpha'_{5,9} + \frac{1}{2} \cdot \alpha'_{4,5} + \frac{3}{4} \cdot \alpha'_{5,11} \quad + \quad \frac{3}{4} \cdot \alpha'_{5,13} + \frac{1}{2} \cdot \alpha'_{4,7} + \frac{1}{4} \cdot \alpha'_{5,15},$$

mit

$$\alpha'_{l,i} := \alpha_{l,i} \cdot 2^{-l+1}/2.$$

Der Faktor, mit dem ein $\alpha'_{l,i}$ einfließt, wird im folgenden Gewicht genannt. Da die Ansatzfunktion die Höhe 1 besitzt, ist der Wert eines Gewichtes der Funktionswert der Ansatzfunktion am Gitterpunkt. Die Basisfunktionen $\phi_{5,9}$ und $\phi_{5,11}$ haben bezüglich $\phi_{4,5}$ jeweils das Gewicht 1/2, entsprechend $\phi_{4,5}$ und $\phi_{4,7}$ bezüglich $\phi_{3,3}$. Die Funktion $\phi_{4,5}$ muss jedoch den Wert des rechten Kindes, $\alpha'_{5,11}$, stärker gewichtet an den Elternknoten emporreichen, während $\phi_{4,7}$ das linke Kind stärker gewichten muss. Es reicht somit nicht aus, nur einen Wert an den hierarchisch höheren Knoten weiterzugeben. Es muss vielmehr sowohl der Fall berücksichtigt werden, dass an späterer Stelle im Baum nach rechts aufgestiegen werden könnte, als auch der Fall, dass nach links aufgestiegen werden könnte.

Es genügt, wenn jeder Knoten das eigene und das Gewicht aller hierarchisch niedrigeren Knoten für beide Fälle an den Elternknoten weitergibt. Im Beispiel aus Abbildung 5.11 geben dann $\phi_{5,9}$ und $\phi_{5,11}$ ihren Wert multipliziert mit ihrem Gewicht, 1/2, jeweils für einen Aufstieg links und rechts an den Elternknoten $\phi_{4,5}$ weiter. Dieser kann den eigenen Einfluss und den der Kinder auf den Elternknoten unabhängig von der Richtung des Aufstiegs berechnen zu $1/2 \cdot (f_{ml} + f_{mr})$ zuzüglich einhalb mal den Wert des eigenen Koeffizienten. Dabei ist f_{ml} der Wert für den Aufstieg nach rechts des linken Kindes, f_{mr} der Wert für den linken Aufstieg des rechten Kindes. Die beiden neuen Werte für den Aufstieg nach links bzw. rechts ergeben sich, indem der Wert für den linken Aufstieg des linken Kindes bzw. den rechten Aufstieg des rechten Kindes addiert wird. Abbildung 5.12 zeigt diesen Prozess für den Aufstieg ab Level 4.

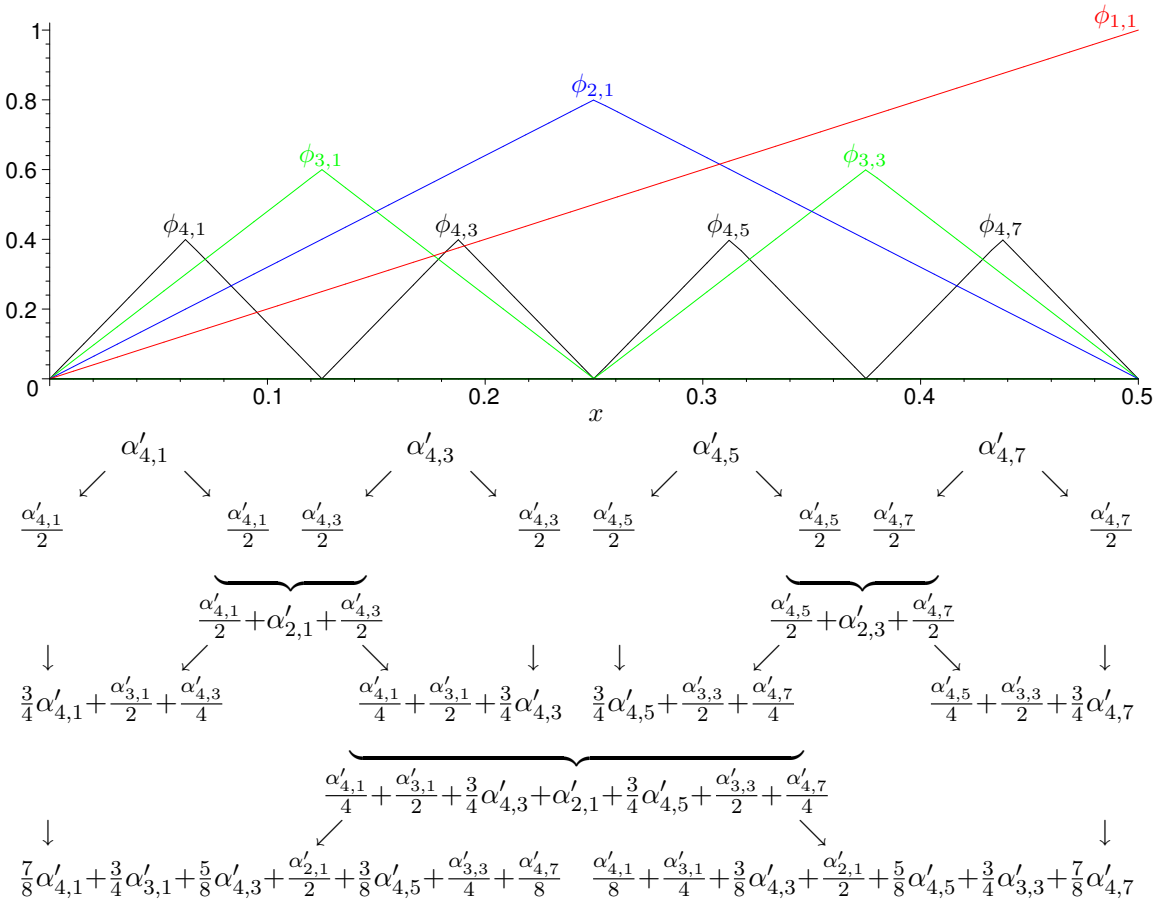


Abbildung 5.12: Berechnung und Weitergabe der Koeffizienten beim *up*.

Das L_2-Skalarprodukt ergibt sich dann in jedem Knoten zu

$$(f_{ml} + f_{mr}).$$

Beim *down*-Schritt wird in einem Knoten $\phi_{l,i}$ der neue linke bzw. rechte Wert zur Weitergabe berechnet zu $f + (f_l + f_r)/2$, wobei f der aktuelle Wert des Knotens selbst ist. Es ist, abgesehen von der Diagonalen, $(C^{\text{up}})^T = C^{\text{down}}$. Für den *up*-Schritt sollte daher die Rückrichtung gelten. Und tatsächlich berechnet sich der neue linke Wert zu $f_l + f/2$ und der neue rechte Wert zu $f_r + f/2$.

Listing 5.4 zeigt die Realisierung des *up*-Schrittes für die Bilinearform $\tilde{a}(\,.\,,.\,)$ in Richtung `dim`, die die rekursiven Baumtraversierungen initialisiert. Der einzige Unterschied zur `down`-Prozedur in Listing 5.2 ist, dass die linke und rechte Grenze als Referenzparameter und nicht als Werteparameter an die Prozedur zur rekursiven Baumtraversierung übergeben werden.

```
up := proc(dim::integer, level::integer, Alpha::Vector, lbf::list,
           indexhash::table)
  # dim       ... Dimension, in der down ausgeführt werden soll
  # level     ... Maximales Level
  # Alpha     ... Koeffizientenvektor
  # indexhash ... Hashfunktion [(Level, Index)*] -> Index der d-dim. Basisfkt
  #
  ## Rückgabe:
  # Beta      ... Ergebnisvektor
  local d, lbf_dminus1, bf, Beta, fl, fr;

  # linke und rechte Grenze
  fl := 0: fr := 0:

  # Dimension des Merkmalraums ermitteln:
  d := nops(lbf[1])/2;

  # Ergebnisvektor Beta erstellen:
  Beta := Vector(Dimension(Alpha));

  # Liste der Basisfunktionen im (d-1)-dimensionalen Raum:
  lbf_dminus1 := li_liste_nb(d-1, level);

  # für jede (d-1)-dimensionale Basisfunktion in Dimension dim im Baum absteigen
  for bf in lbf_dminus1 do
    up_rec(fl, fr, level-add(bf[2*i-1], i=1..d-1)+d-1, 1, 1,
           bf[1..2*(dim-1)], bf[2*(dim)-1..2*(d-1)], Alpha, Beta, indexhash);
  end do;

  # Rückgabe:
  return Beta;

end proc:
```

Listing 5.4: Prozedur für $up_{\texttt{dim}}$.

Die Realisierung der Prozedur `up_rec` zur rekursiven Baumtraversierung für $up_{\texttt{dim}}$ im Paket `DG` ist in Listing 5.5 abgebildet. Die Grundstruktur ist symmetrisch zur Prozedur `down_rec` in Listing 5.3. Die Referenzparameter `fl` und `fr` werden verwendet, um die neue linke bzw. rechte Grenze an die aufrufende Prozedur zurückzugeben.

```
up_rec := proc(fl::evaln, fr::evaln, level::integer,
               akt_level::integer, akt_ind::integer,
               praefix::list, postfix::list,
               Alpha::Vector, Beta::evaln(Vector), indexhash::table)
  # fl, fr             ... Funktionswert an der linken (rechten) Intervallgrenze
  # level              ... Maximales Level
  # akt_level, akt_ind ... Level und Index der akt. Basisfunktion (für Rekursion)
  # praefix, postfix   ... [(Level, Basisfunktion)*] der vorigen
  #                        (nachfolgenden) Dimensionen
  # Alpha              ... Koeffizientenvektor
  # Beta               ... Ergebnisvektor
  # indexhash          ... Hashfunktion [(Level, Index)*] -> Index der Basisfkt.

  local fml, fmr, li_index;
  # Gesamtindex der aktuellen Basisfunktion bestimmen
  li_index := indexhash[[op(praefix), akt_level, akt_ind, op(postfix)]]:

  # neuer mittlerer Wert (links und rechts)
  fml := 0; fmr := 0;

  # Abbruchbedingung (weiterer Abstieg?):
  if (akt_level < level) then

    # Abstieg links
    up_rec(fl, fml, level, akt_level+1, akt_ind*2-1,
           praefix, postfix, Alpha, Beta, indexhash);

    # Abstieg rechts
    up_rec(fmr, fr, level, akt_level+1, akt_ind*2+1,
           praefix, postfix, Alpha, Beta, indexhash);

  end if;

  fl := eval(fl) + (fml + fmr + Alpha[li_index]/2^(akt_level-1)/2) / 2;
  fr := eval(fr) + (fml + fmr + Alpha[li_index]/2^(akt_level-1)/2) / 2;

  # Interpolant * Testfunktion
  Beta[li_index] := (fml + fmr);

end proc:
```

Listing 5.5: Rekursive Baumtraversierung für $up_{\texttt{dim}}$.

Überprüfung der Korrektheit

Die Korrektheit der vorgestellten Realisierungen von *up* und *down* lässt sich im Maple-Paket `DG` durch die Flexibilität der realisierten Testplattform einfach überprüfen. Zum einen kann die Steifigkeitsmatrix C zum d-dimensionalen Dünngitterraum explizit aufgestellt und mit einem Vektor $(\alpha^j)_{1 \leq j \leq N}$ symbolisch multipliziert werden. Zum anderen kann das *up* und *down*-Verfahren auf den selben Vektor $\boldsymbol{\alpha}$ angewendet werden. Durch den direkten Vergleich der Ergebnisse kann die Korrektheit für alle Kombinationen von Level und Dimension sichergestellt werden, für die das explizite Aufstellen der Steifigkeitsmatrix C, limitiert durch die Ressourcen Speicherplatz und Zeit, möglich ist.

5.4 Evaluation der gelernten Funktion

Das Trainieren eines Klassifikators auf die vorklassifizierten Trainingsdaten `X_train` und `Y_train` zum Dünngitterraum $V^{(1)}_{\texttt{level}}$ mit Regularisierungsparameter $\lambda =$ `lambda` erfolgt mittels

```
[> res := classify(level, lambda, X_train, Y_train):
```

Die Funktion `classify` liefert eine Liste mit drei Werten als Rückgabe: die gelernte Funktion, eine Liste der Gitterpunkte und den Vektor der gelernten Koeffizienten $\boldsymbol{\alpha}$.

Die Güte eines Klassifikators wird bezüglich eines Datensatzes gemessen. Für diese Daten muss ebenfalls die Klassenzugehörigkeit bekannt sein. Gemessen wird meist die Genauigkeit des Klassifikators, also das Verhältnis

$$\frac{\text{Anzahl korrekt klassifizierter Daten}}{\text{Anzahl aller Daten}}.$$

Die gelernte Funktion wird hierfür auf jedes Datum angewendet, die Klassenzugehörigkeit bestimmt und mit dem tatsächlichen Klassenlabel verglichen. Wird die Genauigkeit bezüglich der Trainingsdaten gemessen, so spricht man von Trainingsgenauigkeit (*training correctness*). Die Genauigkeit bezüglich eines Datensatzes, der nur zu Testzwecken und nicht zum Training verwendet wird, wird mit Testgenauigkeit (*testing correctness*) bezeichnet.

Im Paket `DG` berechnet

```
[> evaluate_ml(res[1], X_test, Y_test, Z_test):
```

die Genauigkeit für die Daten `X_train` und `Y_train` zum Klassifikator `res[1]`. Der optionale vierte Parameter, `Z_test`, ist ein Vektor der Länge M. Für jedes falsch klassifizierte Datum wird der entsprechende Eintrag in `Z_test` um eins erhöht. Dies ermöglicht genauere Aussagen über die Klassifikation.

Je mehr Trainingsdaten zur Verfügung stehen, desto mehr Information kann während des Trainingsprozesses genutzt werden, um die gesuchte Funktion so gut wie möglich darzustellen. Es liegt daher nahe, alle vorhandenen klassifizierten Daten für den Lernvorgang zu verwenden. Allerdings ist der Klassifikator für die Trainingsdaten optimiert und liefert auf die Grundgesamtheit der Daten unter Umständen schlechtere Ergebnisse. Dieses Phänomen nennt sich Überanpassung (overestimation). Die Trainingsgenauigkeit ist zur Beurteilung eines Klassifikators nicht aussagekräftig genug, da hierfür Trainings- und Testmenge identisch und damit nicht unabhängig sind. Sie ist zwar ein guter Indikator, ob der Klassifikator mächtig genug ist, das vorliegende Problem zu lernen. Sie sagt allerdings nichts über seine Generalisierungsfähigkeiten im Hinblick auf unbekannte Daten aus.

Daher bedient man sich zur Bewertung eines Klassifikators der sogenannten *hold-out*-Methode. Ein Teil der Daten wird zu Testzwecken zurückgehalten; die Daten werden in disjunkte Mengen von Trainings- und Testdaten unterteilt. Die Auswertung auf die Trainingsdaten liefert ein Maß für die Güte der Vorhersage für unbekannte Daten.

Dies widerspricht jedoch dem Wunsch, möglichst viel Information für den Lernvorgang zu nutzen. Zudem sind in vielen realistischen Szenarios nur für wenige Daten die Klassenlabel bekannt. Hauptgrund hierfür ist, dass die Zuordnung zu den Klassen meist manuell geschehen muss. Die Unterteilung in gleichgroße Test- und Trainingsmengen ist nicht anwendbar, wenn die Teilmenge zum Lernen nicht mehr repräsentativ für die Gesamtheit der Daten ist, d. h. wenn sie nicht ausreichend Information über das zu lernende Problem beinhält. Stattdessen kann die Technik der Kreuzauswertung (*cross-validation*) verwendet werden.

5.4.1 Kreuzauswertung

Die k-fache Kreuzauswertung (k-fold cross-validation) verwendet alle Daten sowohl zum Lernen als auch zum Testen. Sie nutzt die vorhandenen Daten in dieser Hinsicht optimal aus.

Die Gesamtmenge der Daten S wird unterteilt in k gleich große, disjunkte Teilmengen. Nun wird k-mal der Klassifikator auf jeweils $k-1$ Teilmengen trainiert. Abbildung 5.13 zeigt dieses Schema für $k = 4$. Für die verbleibende Teilmenge wird die Testgenauigkeit bestimmt. Auf jeden der Datenpunkte wird einmal getestet und $(k-1)$-mal trainiert. Die k erhaltenen Testgenauigkeiten werden kombiniert. Bei gleicher Größe der Teilmengen ist die Testgenauigkeit auf alle Daten das arithmetische Mittel. Bei ungleicher Größe müssen die Teilwerte entsprechend gewichtet werden.

Teilmenge 1	Teilmenge 2	Teilmenge 3	Teilmenge 4

Trainingsdaten			Testdaten
Trainingsdaten		Testdaten	Trainingsdaten
Trainingsdaten	Testdaten	Trainingsdaten	
Testdaten	Trainingsdaten		

Abbildung 5.13: Vierfache Kreuzauswertung.

Anzumerken ist, dass die Zuteilung zu den Teilmengen zufällig geschehen sollte. Außerdem sollte möglichst jede der Teilmengen gleich aussagekräftig sein. Zweiteres wird von der stratifizierten Variante der Kreuzauswertung unterstützt: Die zusätzliche Forderung an die Aufteilung der Daten ist, dass in jeder Teilmenge der Anteil der Daten jeder Klasse in etwa gleich ist.

Wird dieses Prinzip mit maximalem k eingesetzt, so wird $(M-1)$-mal auf $M-1$ Daten trainiert und auf das verbliebene Datum getestet. Man spricht dann von n-facher Kreuzauswertung (*n-fold cross-validation* oder *hold-one-out cross-validation*). Allerdings ist diese Variante sehr aufwändig. In [Koha95] wird die in der Praxis gängige Faustregel der zehnfachen Kreuzauswertung untermauert: Für Genauigkeitsabschätzungen wird eine stratifizierte zehn- bis zwanzigfache Kreuzauswertung empfohlen.

Im Paket `DG` kann mit

```
[> n_fold_cv(k, level, lambda, X, Y, Z);
```

die `k`-fache Kreuzauswertung für die Daten `X` und `Y` im Dünngitterraum $V^{(1)}_{\texttt{level}}$ mit Regularisierungsparameter $\lambda =$ `lambda` berechnet werden. Rückgabewerte sind Trainings- und

Testgenauigkeit. In dem optionalen Parameter `Z` wird analog zur Verwendung in der Prozedur `evaluate_ml` für jedes missklassifizierte Testdatum der entsprechende Eintrag um eins erhöht.

Für die Kombinationstechnik steht zur Kreuzauswertung die Prozedur

```
[> n_fold_cv_combi(k, level, lambda, X, Y, Z);
```

zur Verfügung. Für die zehnfache Kreuzauswertung gibt es die Abkürzungen `ten_fold_cv` und `ten_fold_cv_combi`. Mit

```
[> set_random_function(f);
```

kann eine eigene Funktion zur Generierung einer (zufälligen) Permutation der Indizes der Datenpunkte gesetzt werden.

Das Paket `DG` enthält eine Reihe zusätzlicher Hilfsmittel zur grafischen Evaluierung, beispielsweise zur Darstellung der gelernten Funktion oder eines Datensatzes inklusive der Markierung missklassifizierter Datenpunkte. Allerdings lässt sich eine gelernte Funktion nur zu einem Merkmalsraum mit maximal zwei Dimensionen auf natürliche Art und Weise darstellen. Funktionen zur Speicherung gewonnener Bilder in gängigen Grafikformaten runden das Paket `DG` ab.

6 Testfälle und Ergebnisse

Das Maple-Paket `DG` wird nun auf drei binäre Klassifikationsprobleme angewendet. Es handelt sich hierbei um drei Testbeispiele, die verschiedene Charakteristika von Klassifikationsproblemen abdecken und die Hinweise auf die Verwendbarkeit der Dünngittertechnik zur Klassifikation geben. Schwerpunkt der Betrachtungen ist insbesondere der Einfluss des Glattheitsarguments, d. h. des Regularisierungsparameters λ. Ein weiterer Schwerpunkt liegt auf dem Vergleich zwischen der direkten Finite-Elemente-Dünngittertechnik und der Kombinationstechnik (Kapitel 4.4).

Der Merkmalsraum wird für die folgenden Beispiele auf $[0,1]^d$ normalisiert. Die Werte für die beiden Klassen sind $+1$ und -1. Die Klassenzugehörigkeit ändert sich am Nulldurchgang des Klassifikators: Datenpunkte mit Funktionswerten größer oder gleich null werden zur positiven Klasse gezählt.

Als Basis für die direkte Dünngittertechnik wird die in Kapitel 4.3 vorgestellte Variante mit ausgeklappten Basisfunktionen und zusätzlichen Basisfunktionen auf dem Rand verwendet. Grund hierfür ist die bessere Vergleichbarkeit mit der Kombinationstechnik: Das zugrundeliegende Gitter entspricht dem Gitter der Kombinationstechnik auf dem selben Level. Für die Klassifikationsaufgaben in Kapitel 6.1 und Kapitel 6.3 werden für die direkte Dünngittertechnik auch ohne Basisfunktionen auf dem Rand ähnlich gute Werte erzielt.

6.1 Schachbrett-Datensatz

Das erste Beispiel ist ein 4×4-Schachbrettmuster (Abbildung 6.1). Es handelt sich um 1000 mehr oder weniger gleichverteilte Datenpunkte, deren Klassenzugehörigkeiten anhand von ihrer Lage festgelegt werden. Aufgabe des Klassifikators ist, diese regelmäßige Struktur zu lernen. Klassifikationsalgorithmen wie einfache Entscheidungsbäume, die auf Entscheidungen für einzelne Attribute basieren, haben mit derartigen Datensätzen Probleme: Für jeden Wertebereich eines Attributes gibt es in etwa gleich viele Datenpunkte aus beiden Klassen.

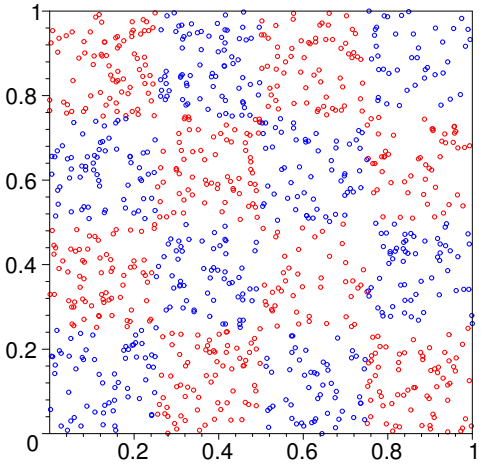


Abbildung 6.1: 4×4-Schachbrettmuster. 1000 Datenpunkte.

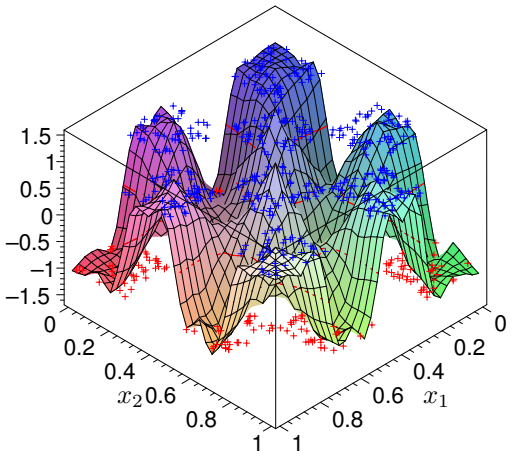


Abbildung 6.2: Trainingspunkte und gelernte Funktion auf Level 5 für $\lambda = 0.001$.

Im Gegensatz zu den meisten realistischen Anwendungen fehlt Rauschen in den Daten: Es gibt keine Datenpunkte mit falscher Klassenzugehörigkeit. Charakteristisch ist jedoch, dass die Lage der Punkte, d. h. die Auswertungsstellen der zu approximierenden Funktion, zufällig ist. Im Hinblick auf Auswertungen von Klassifikatoren muss jedoch beachtet werden, dass zufallsbedingt ein anderer Schachbrettdatensatz andere Werte liefern kann. Bei 1000 Punkten sind jedoch nur geringfügige Abweichungen zu erwarten.

Annahme für die folgenden Betrachtungen ist, dass keine weiteren Datenpunkte zur Verfügung stehen. Zur Untersuchung des Einflusses des Regularisierungsparameters λ wurden daher mittels zehnfacher Kreuzauswertung die Trainings- und Testgenauigkeiten für verschiedene Werte von λ bestimmt. Abbildung 6.3 zeigt den Verlauf der Genauigkeitswerte für die Level vier bis neun bei Verwendung der direkten Dünngittertechnik. Der Bereich $\lambda < 7.94 \cdot 10^{-8}$ (senkrechte Linie), in dem praktisch keine Regularisierung statt findet, wird dabei durch einen weiteren Datenpunkt mit $\lambda = 10^{-16}$ angedeutet, der aus Platzgründen an der Stelle 10^{-8} aufgetragen ist.

Für beide Kurvenverläufe ist deutlich sichtbar, dass die Genauigkeitswerte für große Werte von λ gegen 50% streben: Ein zu groß gewählter Wert für λ erzwingt zu große Glattheit. Die gelernte Funktion ist nahezu konstant. Für $\lambda > 1/2$ ist der Klassifikator entweder auf dem gesamten Gebiet positiv oder auf dem gesamten Gebiet negativ, und alle Datenpunkte werden der selben Klasse zugeteilt.

Die Klassifikatoren für Level vier und fünf erreichen keine Trainingsgenauigkeit von 100%. Die Klassifikatoren sind nicht mächtig genug, um die zugrundeliegende Struktur zu lernen: Die Auflösung des Gitters ist nicht fein genug, siehe Abbildung 6.4.

Für Level kleiner oder gleich vier enthält das dünne Gitter keine Gitterpunkte (feine schwarze Punkte) innerhalb der mittleren vier Quadrate, siehe auch Abbildung 4.8 für ein dünnes Gitter mit Level vier. Die nächstliegenden Gitterpunkte befinden sich genau auf dem Rand der Klassengrenzen des zugrundeliegenden Schachbrettmusters. Erst auf Level fünf könnte durch entsprechende Wahl von nur 16 Basisfunktionen (nämlich die aus $W_{(3,3)}$) eine exakte Klassifikation erreicht werden. Da die Trainingsdaten der beiden Klassen nicht gleich weit von den zugrundeliegenden Trennlinien des Schachbrettmusters entfernt liegen, werden auch Basisfunktionen auf den Rändern der vier mittleren Quadrate zur genaueren Anpassung verwendet. Diese verfälschen das Schachbrettmuster. Ab Level sechs können die Trainingsdaten nahezu vollständig der richtigen Klasse zugeordnet werden.

Für die Testgenauigkeiten auf Level vier und fünf ist für kleiner werdendes λ bis etwa 0.001 ein Anstieg der Genauigkeitswerte zu beobachten. Anschließend sinkt die Genauigkeit wie-

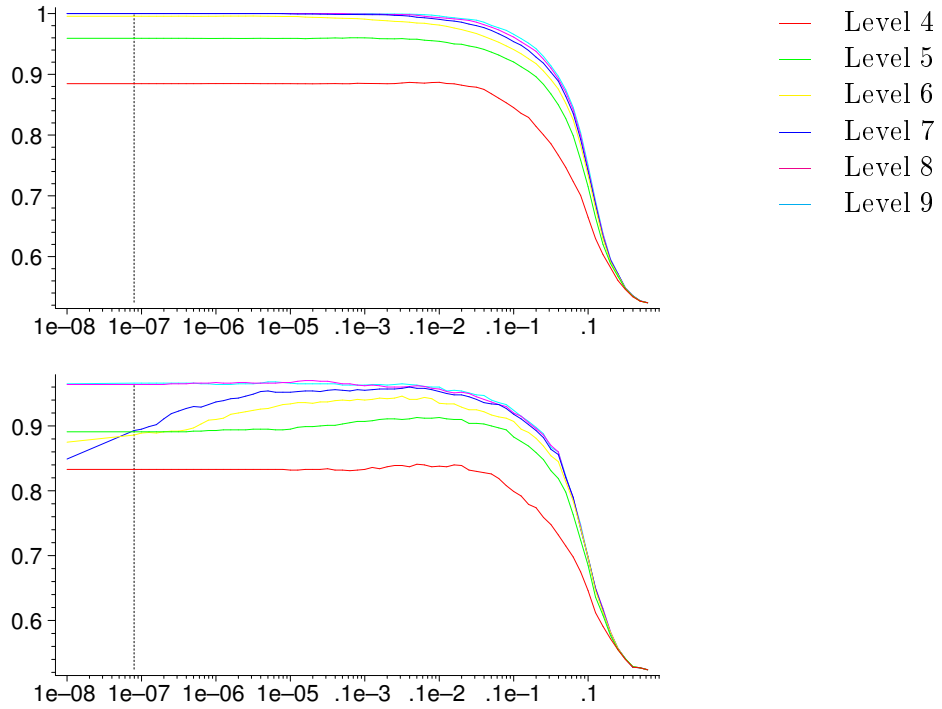


Abbildung 6.3: Trainings- (a) und Testgenauigkeit (b) in Abhängigkeit von λ (logarithmische Skala) für 10-fache Kreuzauswertung, direkte FE-Methode.

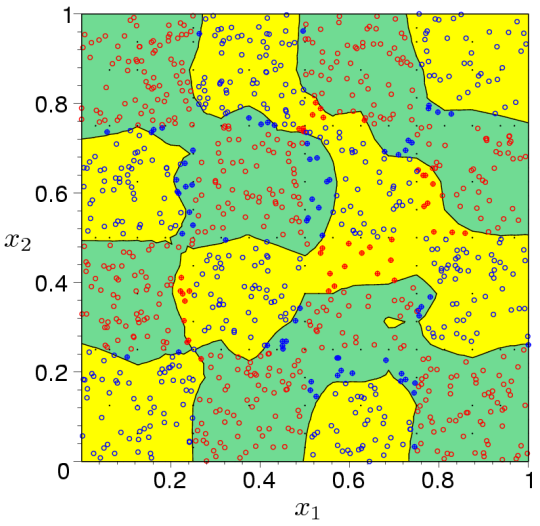
der: Die Glattheitsanforderung ist zu gering, es tritt Überanpassung auf. Der Klassifikator passt sich zu genau den Trainingsdaten an und generalisiert nicht mehr gut genug. Beide Kurven brechen nicht stärker ein, da die dünnen Gitter auf Level vier und fünf das Gebiet nicht feiner auflösen können: Die Klassifikatoren besitzen aufgrund der niedrigen Auflösung ihres Funktionsraumes eine gewisse Grundglattheit.

Im Allgemeinen ist zu erwarten, dass mit steigendem Level ein zunehmend stärkerer Einbruch für sehr kleine Werte von λ zu erwarten ist. Dies trifft zu für die Level sechs und sieben. Level sieben erzielt eine höhere maximale Genauigkeit. Bei fehlender Glattheitsanforderung ist der Klassifikator jedoch über zwei Prozent schlechter als auf Level sechs. Aus welchen Gründen und durch welche Nebeneffekte dies nicht für die Level acht und neun zutrifft, konnte mit bisherigen Experimenten noch nicht erklärt werden.

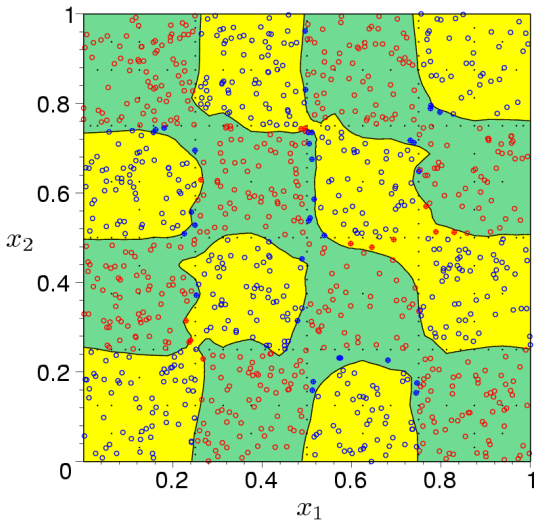
Abbildung 6.5 verdeutlicht diese Überlegungen für verschiedene Werte von λ auf Level sieben. Die Klassifikatoren wurden unter Verwendung der Kombinationstechnik erstellt. Bei dieser ist der der Effekt der Überanpassung deutlicher zu beobachten. Die folgenden Betrachtungen treffen jedoch qualitativ auch für die direkte Dünngittertechnik zu.

Bei großem λ ist die gelernte Funktion so glatt, dass sie nur an wenigen Stellen einen Nulldurchgang besitzt. Für $\lambda = 0.1$ wird bereits in jedem Quadrat ein Klassenunterschied erreicht. Für $\lambda = 0.001$ wird das Schachbrettmuster erkannt. Der genaue Verlauf der Klassifikationsgrenzen ist auf die zufallsbedingte Lage der Datenpunkte zurück zu führen. Bei weiterer Reduzierung der Glattheitsbedingung ist die Grenze zwischen den beiden Klassen nicht mehr glatt und wird zu genau aufgelöst. Wird die Glattheitsanforderung nahezu vernachlässigt, so kann an vielen Stellen in Bereichen, in denen keine Trainingsdaten vorliegen, ein Überspringen der gelernten Funktion beobachtet werden.

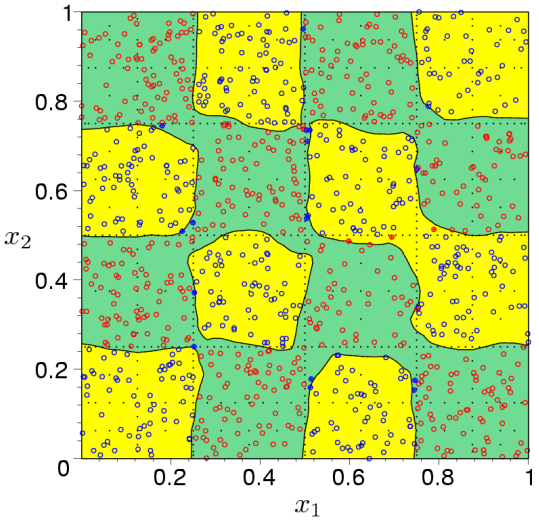
Abbildung 6.6 zeigt für die Kombinationsmethode den Verlauf der Trainingsgenauigkeit bei zehnfacher Kreuzauswertung für die selben Werte des Regularisierungsparameters wie bei Verwendung der direkten FE-Technik in Abbildung 6.3. Es ergibt sich ein ähnliches



Level 4



Level 5



Level 6

Abbildung 6.4: Klassifikation für verschiedene Level, $\lambda = 0.001$, FE-Methode.

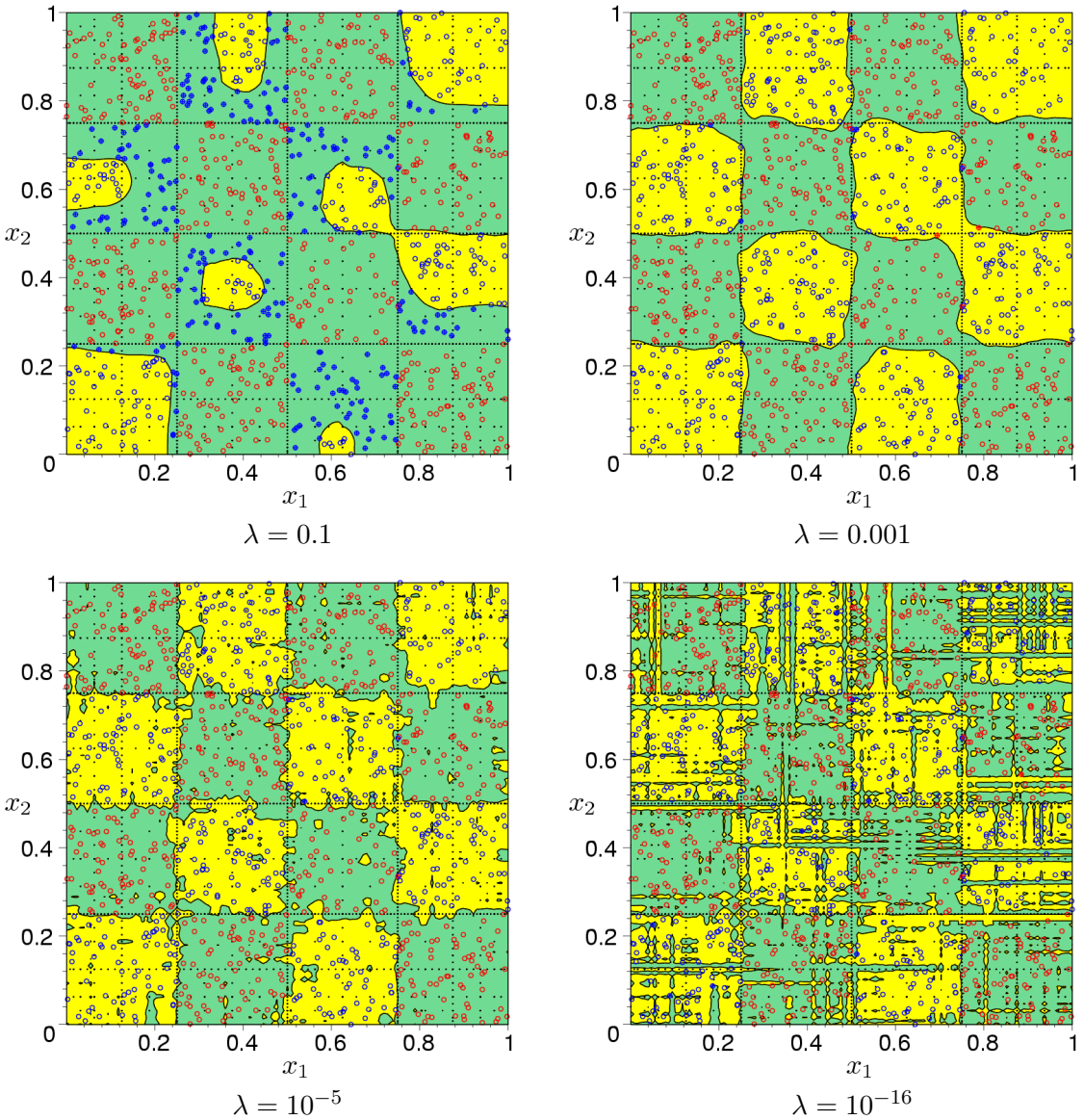


Abbildung 6.5: Überanpassungseffekte am Beispiel von Level 7 für verschiedene Werte von λ, Kombinationsmethode.

Gesamtbild. Interessanterweise kann jedoch für kleiner werdende λ nach Erreichen des maximalen Wertes eine Abnahme der Genauigkeitswerte beobachtet werden: Das Lernen von Teilklassifikatoren auf unterschiedlichen Gittern wirkt sich leicht negativ auf die Güte des Gesamtklassifikators aus. Im Vergleich mit der direkten Dünngittertechnik werden um ein bis zwei Prozent schlechtere Werte für die Trainingsdaten erzielt.

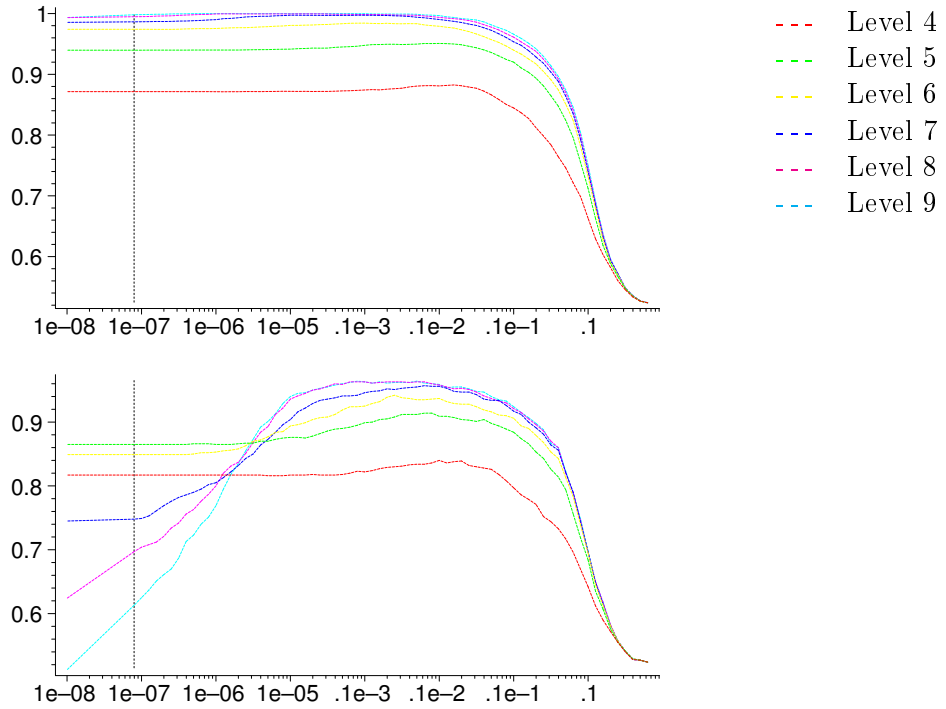


Abbildung 6.6: Trainings- (a) und Testgenauigkeit (b) in Abhängigkeit von λ (logarithmische Skala) für 10-fache Kreuzauswertung, Kombinationsmethode.

Für die Testgenauigkeit zeigt sich im Hinblick auf die Überanpassung das erwartete Verhalten: Die gelernte Funktion auf Level vier erzielt den schlechtesten Maximalwert. Durch die geringe Auflösung ihres Merkmalraumes ist sie jedoch so glatt, dass sich bei fehlender Glattheitsanforderung nur ein geringer Überanpassungseffekt einstellt. Je höher das Level ist, desto wichtiger wird die Glattheitsbedingung: Für zu kleine λ werden die Funktionswerte der Trainingsdaten zu genau approximiert und die Generalisierungsfähigkeit schwindet.

Die Kombinationsmethode ist wesentlich anfälliger für die Wahl von zu kleinen Werten für λ als die direkte Methode. Es wird zum Beispiel für erstere auf Level sieben die maximale Genauigkeit von 95.7% für $\lambda = 6.31 \cdot 10^{-4}$, für zweitere 96% für $\lambda = 3.98 \cdot 10^{-4}$ erzielt. Bei einem Spielraum von einem Prozent werden für die direkte FE-Methode Werte zwischen 96% und 95% für λ bis zu $3.98 \cdot 10^{-6}$ erzielt. Bei der Kombinationsmethode erhält man nur bis $\lambda = 1.25 \cdot 10^{-4}$ Genauigkeitswerte zwischen 95.7% und 94.7%.

Tabelle 6.1 stellt die Trainings- und Testgenauigkeiten für n-fache Kreuzauswertung für die Level vier bis acht dar. Der Regularisierungsparameter λ wurde durch zehnfache Kreuzauswertung bestimmt. Es ist jeweils der Wert, für den in den Abbildungen 6.3 und 6.6 die besten Testgenauigkeiten berechnet wurden.

Bis auf die Testgenauigkeit für Level sieben und neun sind alle Werte der direkten Finite-Elemente-Dünngittertechnik besser als die Werte, die mit der Kombinationstechnik erzielt wurden. Für beide Techniken kann ab Level sieben das Schachbrettmuster mit weniger als 4% Fehler rekonstruiert werden.

Level	Genauigkeit FE-Methode			Genauigkeit Komb.-Methode		
	λ	Training	Test	λ	Training	Test
4	0.000501	0.891422	0.844	0.001	0.879881	0.840
5	0.001	0.954261	0.913	0.000794	0.950952	0.911
6	0.000316	0.984042	0.941	0.000251	0.981984	0.932
7	0.000398	0.994014	0.962	0.000631	0.991990	0.962
8	1.58489e-5	1	0.966	0.000631	0.995007	0.963
9	6.30957e-6	1	0.965	0.000501	0.998008	0.965

Tabelle 6.1: Genauigkeitswerte für die direkte FE-Methode und die Kombinationsmethode, n-fache Kreuzauswertung für die besten ermittelten Werte von λ aus der zehnfachen Kreuzauswertung.

6.2 Spiral-Datensatz

Der nächste Datensatz besteht aus zwei ineinander verschlungenen Spiralen (Abbildung 6.7). Er wurde bereits unter anderem von [Sing98] und [GaGT01] verwendet. Jede der beiden Spiralen besteht aus 97 Datenpunkten einer Klasse. Dieser künstliche Datensatz eignet sich gut, um die Fähigkeit von Klassifikationsverfahren zu testen, auch komplexe Strukturen erkennen zu können. Interessanterweise kann ein Nächster-Nachbar-Klassifikator angewendet auf alle Datenpunkte die beiden Spiralen gut voneinander trennen: Die Klassenzuordnung dieses einfachen Verfahrens auf dem Gebiet $[0,1]^2$ entspricht den Flächen eines Voronoi-Diagrammes zu den Datenpunkten. Bei dem Entfernen einzelner Punkte geht für den Nächster-Nachbar-Klassifikator die Spiralstruktur jedoch schnell verloren.

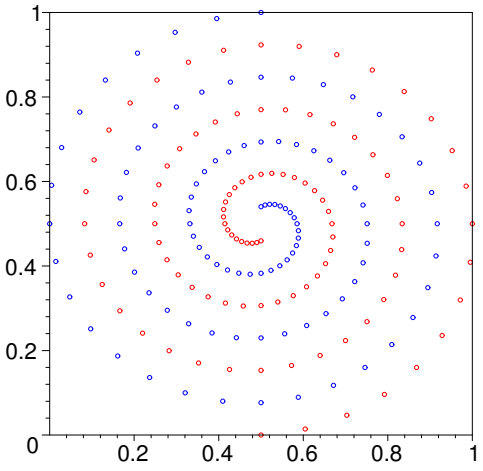


Abbildung 6.7: Spiraldatensatz, 194 Punkte.

Komplexere Verfahren haben jedoch teilweise gravierende Probleme mit dem Lernen der Spiralstruktur. In [Sing98] wurde beispielsweise nachgewiesen, dass manche Neuronale Netze diese komplexe Struktur gar nicht erfassen können. Runde Klassifikationsgrenzen und ineineinander verschachtelte Strukturen stellen für manche Klassifikationsalgorithmen einen besonderen Härtefall dar.

Abbildung 6.8 stellt die Trainings- und Testgenauigkeiten für Level vier bis neun in Abhängigkeit von λ dar. Verwendet wurde wieder zehnfache Kreuzauswertung zur Identifikation des besten Wertes des Regularisierungsparameters. Für die Trainingsgenauigkeit ergibt sich ein ähnliches Bild wie bei dem Schachbrett-Datensatz: Je höher das Level ist, desto besser ist die Genauigkeit. Hier können bereits ab Level fünf die Trainingspunkte vollständig gelernt werden, auch wenn diese nur noch 90% der ursprünglichen Spiraldaten darstel-

len: Die Klassifikation mit dünnen Gittern kann so komplexe Strukturen wie ineinander verschlungene Spiralen lernen.

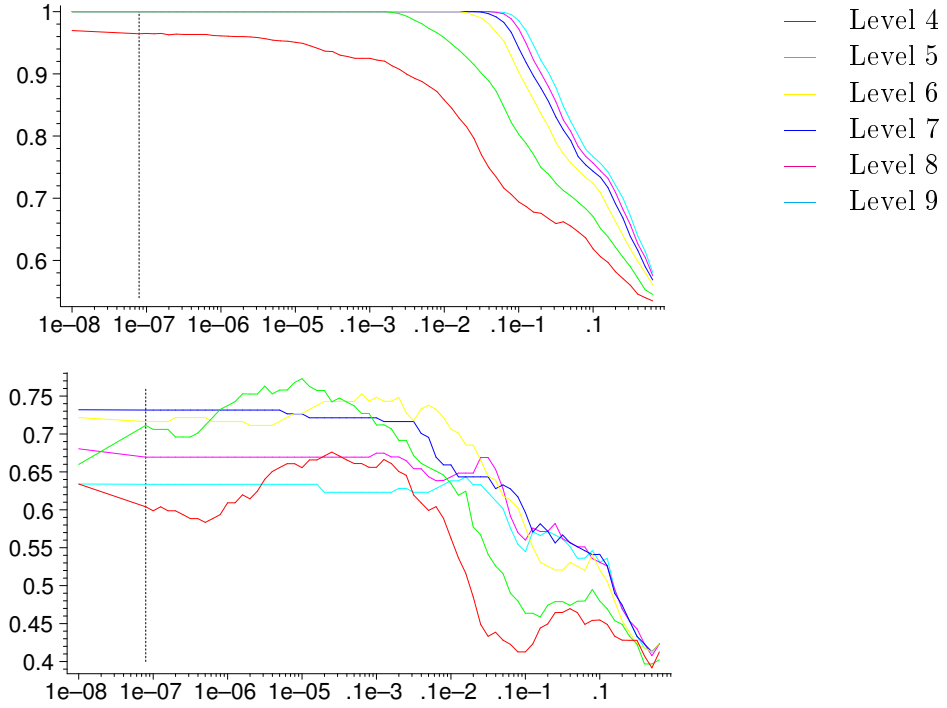


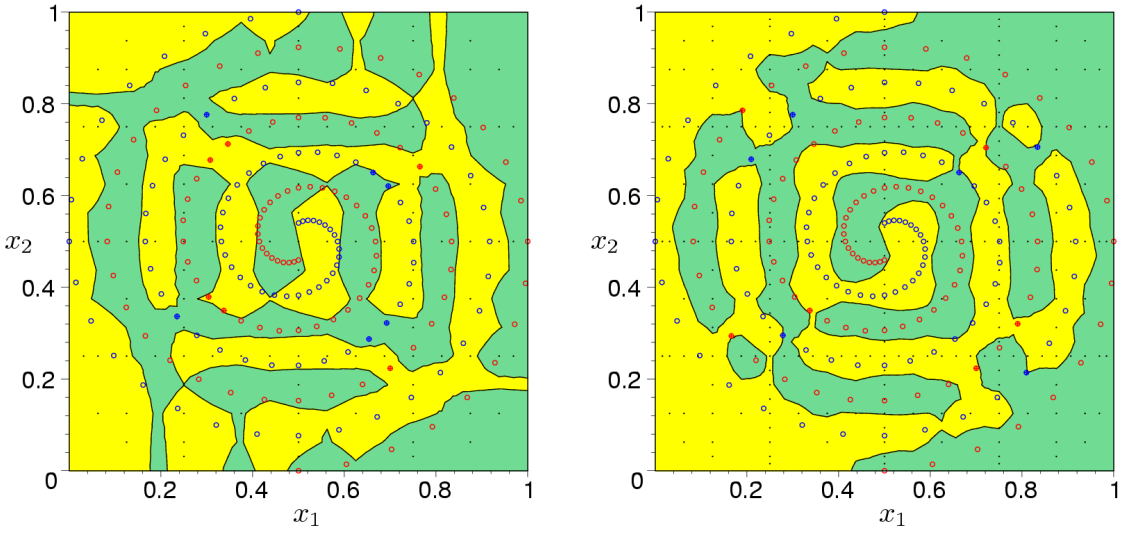
Abbildung 6.8: Trainings- (a) und Testgenauigkeit (b) in Abhängigkeit von λ (logarithmische Skala) für 10-fache Kreuzauswertung, direkte FE-Methode.

Beachtet werden muss, dass bei zehnfacher Kreuzauswertung nicht mehr notwendigerweise ausreichend viele Datenpunkte für den Lernvorgang vorhanden sind, um die Spiralstruktur rekonstruieren zu können. Beim Schachbrett-Datensatz gibt es aus Sicht des Funktionsraumes mehr Daten als Information: Jedes Feld des Schachbrettes enthält im Schnitt etwa 62 Datenpunkte. Im Gegensatz dazu trägt hier jeder Punkt einen größeren Teil zur Gesamtstruktur bei. Dies erklärt kleinere Schwankungen in den dargestellten Kurven.

In Abbildung 6.9 werden gelernte Klassenzuordnungen für die Dünngitterräume mit Level vier bis sieben gezeigt. Auf Level vier kann zwar die Klassenzugehörigkeit für die meisten Datenpunkte gelernt werden, aber die eigentliche Spiralstruktur kann nicht dargestellt werden. Stattdessen kann die rechtwinklige Struktur des dünnen Gitters erahnt werden. Fehlen Punkte bei der zehnfachen Kreuzauswertung im Trainingsset insbesondere auf den Diagonalen des Gebietes und am Rand, so werden sie falsch klassifiziert.

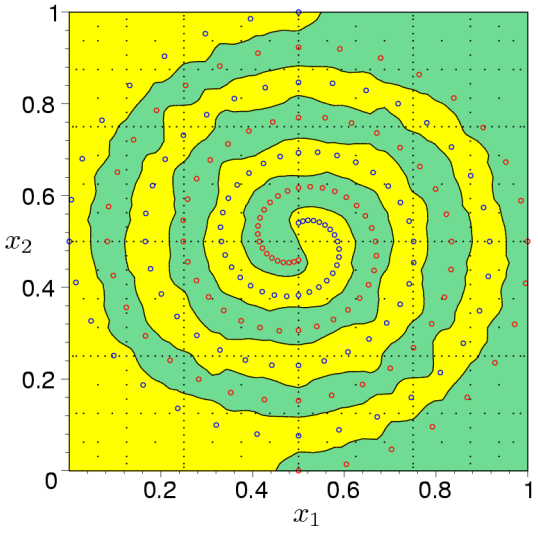
Das gute Abschneiden bei der Trainingsgenauigkeit des Klassifikators zum Dünngitterraum mit Level fünf erklärt sich wieder durch die begrenzte Auflösung des Merkmalraumes. Es können zwar bereits größere Teile der Spirale erkannt werden als auf Level vier, aber das Gebiet einer Klasse zerfällt weiterhin in mehr oder weniger zusammenhängende Blöcke. Es gibt allerdings noch nicht genügend Basisfunktionen, um beim Fehlen einzelner Datenpunkte in den Trainingsdaten innerhalb der erkannten Blöcke diese zu unterteilen. Punkte am Ende der Blöcke, d. h. auf den Diagonalen des Gebietes, bereiten weiterhin Probleme.

Ab Level sechs kann die Spiralstruktur vollständig richtig gelernt werden und für Level sieben wird die Grenzlinie zwischen den beiden Klassen glatter. Beim Training auf alle Datenpunkte ergibt sich für Level acht und neun ein ähnliches Bild, sofern λ nicht zu klein gewählt wird. Das schlechtere Abschneiden der Klassifikatoren auf den Leveln größer sieben bei der Testgenauigkeit der zehnfachen Kreuzauswertung kann durch die Tatsache erklärt werden, dass jedes Fehlen eines Punktes im Trainingsset sofort gelernt werden kann: Es

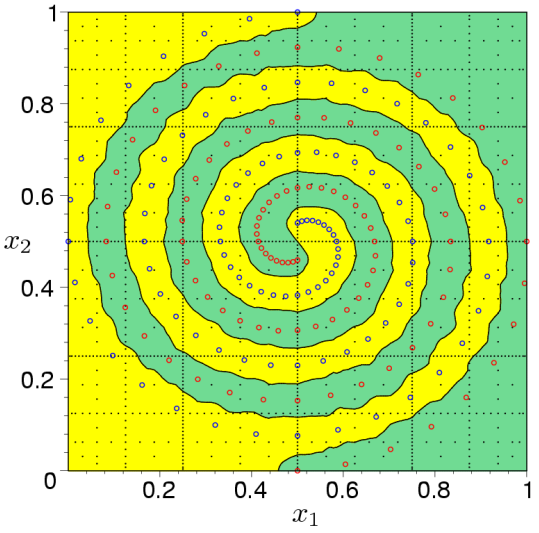


Level 4

Level 5



Level 6



Level 7

Abbildung 6.9: Klassifikationsgrenzen für Level 4 bis 7, direkte FE-Methode.

wird eine stückweise Spirale gelernt, was im Hinblick auf die gewünschte Gesamtstruktur zu Fehlklassifikationen führt.

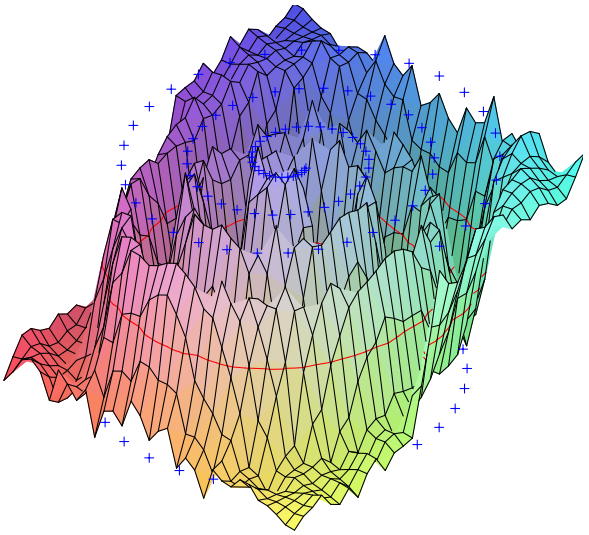


Abbildung 6.10: Trainingspunkte und gelernte Funktion auf Level 7.

Abbildung 6.10 zeigt die Trainingsdaten und die gelernte Funktion für Level sieben. In der Mitte der Spirale gibt es mehr Basisfunktionen als weiter außen: Die Klassenwerte $+1$ und -1 der Datenpunkte können dort trotz Glattheitsanforderung näher erreicht werden, als in den äußeren Bereichen der Spirale. In den Ecken des Gebietes können die Trainingsdaten sehr gut approximiert werden.

Bei Verwendung der Kombinationstechnik fallen die Genauigkeitswerte etwas schlechter aus, als bei Verwendung der direkten FE-Technik. Für die Trainingsgenauigkeit lässt sich auf Level fünf wieder ein leichter Rückgang bei zu kleinen Werten für λ beobachten.

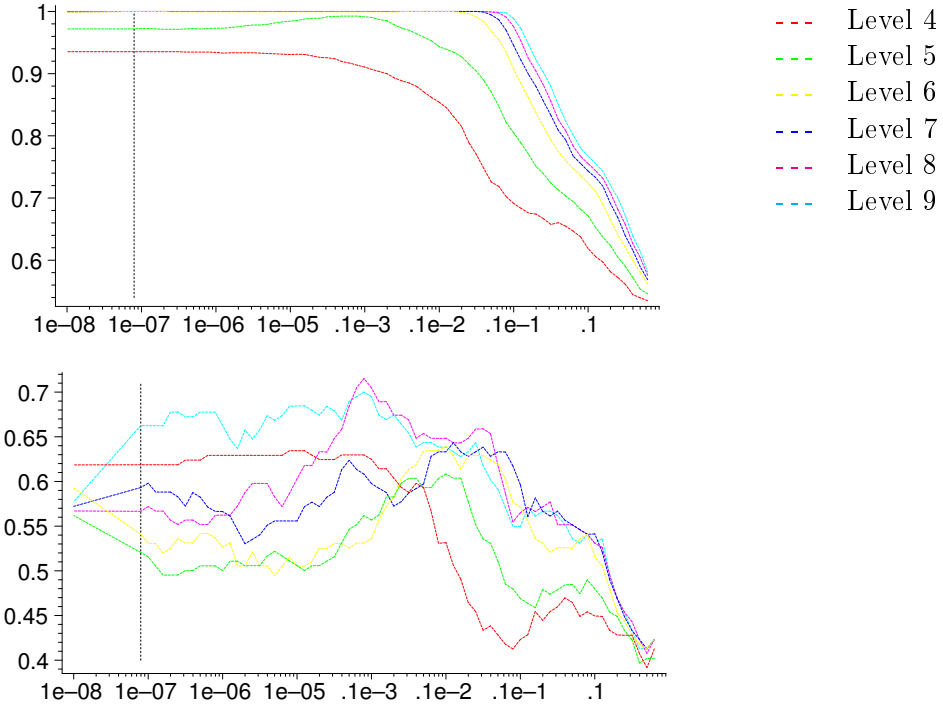


Abbildung 6.11: Trainings- (a) und Testgenauigkeit (b) in Abhängigkeit von λ (logarithmische Skala) für 10-fache Kreuzauswertung, Kombinationsmethode.

Bei Betrachtung der Testgenauigkeiten fällt auf, dass erst bei Level acht der Maximalwert erreicht wird. Im Vergleich hierzu kann bei der direkten Dünngittermethode ein Rückgang der Genauigkeit für Level größer als sieben verzeichnet werden.

Für beide Techniken wurde für jedes Level der für die Testgenauigkeit beste Wert von λ mittels zehnfacher Kreuzauswertung ermittelt. Tabelle 6.2 zeigt zu diesen Werten des Regularisierungsparameters die Trainings- und Testgenauigkeiten für n-fache Kreuzauswertung.

Level	Genauigkeit FE-Methode			Genauigkeit Komb.-Methode		
	λ	Training	Test	λ	Training	Test
4	2.51189e-5	0.928049	0.680412	7.94328e-6	0.927835	0.680412
5	0.00001	1	0.752577	0.001	0.937557	0.628866
6	6.30957e-5	1	0.742268	0.001	1	0.587629
7	5.01187e-6	1	0.783505	0.001259	1	0.639175
8	0.0001	1	0.701031	7.94328e-5	1	0.742268
9	0.001585	1	0.587629	7.94328e-5	1	0.701031

Tabelle 6.2: Genauigkeitswerte für die direkte FE-Methode und die Kombinationsmethode, n-fache Kreuzauswertung für die besten ermittelten Werte von λ aus der zehnfachen Kreuzauswertung.

Die Kombinationsmethode hinkt der direkten FE-Methode bei Trainings- und Testgenauigkeit in etwa um ein Level hinterher: Die komplette Darstellung der Trainingsdaten gelingt erst mit Level sechs. Das Maximum der Testgenauigkeit kann erst auf Level acht und nicht bereits auf Level sieben beobachtet werden. Für beide Verfahren kann nach Erreichen des Maximums ein Rückgang der Testgenauigkeiten beobachtet werden. Ab diesen Stellen kommt die Überanpassung zu tragen: Fehlende Punkte im Trainingsset werden zunehmend falsch klassifiziert. Es wird eine Spirale „mit Einbuchtung“ gelernt.

Die beste Testgenauigkeit von 78.35% wurde mit der direkten Dünngittertechnik auf Level sieben für $\lambda = 5.01187 \cdot 10^{-6}$ erreicht. In [Sing98] wurde eine maximale Genauigkeit von 77.2% berichtet.

6.3 Ripley-Datensatz

Der dritte Datensatz, auch bekannt als Ripley-Datensatz, entstammt [RiHj95] (Abbildung 6.12). Es handelt sich um einen synthetisch generierten Datensatz mit 250 Trainingspunkten und 1000 Testpunkten. Dieser Datensatz deckt eine weitere Eigenschaft realistischer Datensätze ab: Er enthält einen größeren Anteil Rauschen.

Es werden wieder mittels zehnfacher Kreuzauswertung auf die Trainingsdaten die besten Werte für den Regularisierungsparameter λ ermittelt. Abbildung 6.13 zeigt den Verlauf der Genauigkeitswerte für die direkte Dünngittertechnik, Abbildung 6.14 den Verlauf für die Kombinationstechnik.

Interessant ist, dass sich die Testgenauigkeit für große Werte von λ nicht wie in den vorherigen Beispielen der 50%-Grenze annähert. Die Anzahl der Daten von beiden Klassen ist zwar wieder gleich groß, die zu lernende Struktur jedoch wesentlich einfacher: Bereits durch das Lernen einer leicht abschüssigen Ebene können die beiden Klassen verhältnismäßig gut getrennt werden.

Bei selbem Level kann mit der direkten Dünngittertechnik eine bessere Trainingsgenauigkeit erzielt werden, als mit der Kombinationstechnik. Bei ersterer können die Trainingsdaten inklusive Rauschen bereits ab Level sechs, bei zweiterer erst ab Level acht fehlerfrei

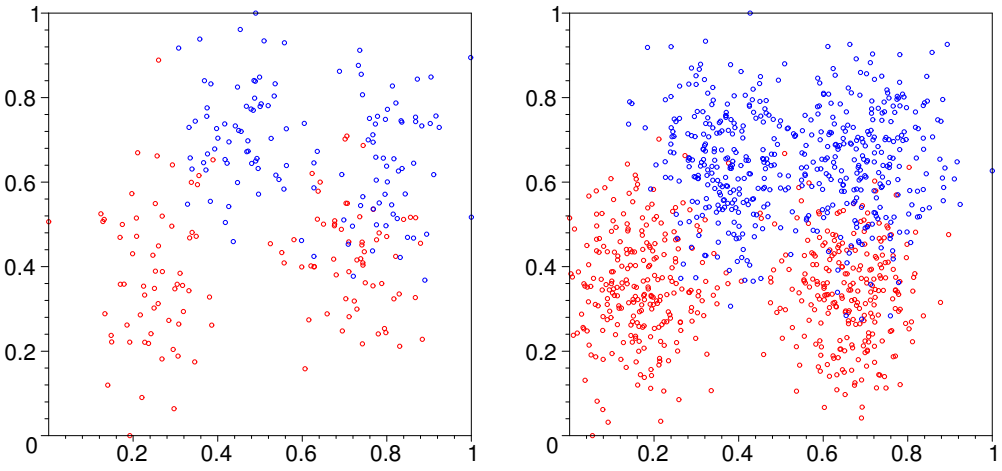


Abbildung 6.12: Ripley-Datensatz. 250 Trainingspunkte (a), 1000 Testpunkte (b).

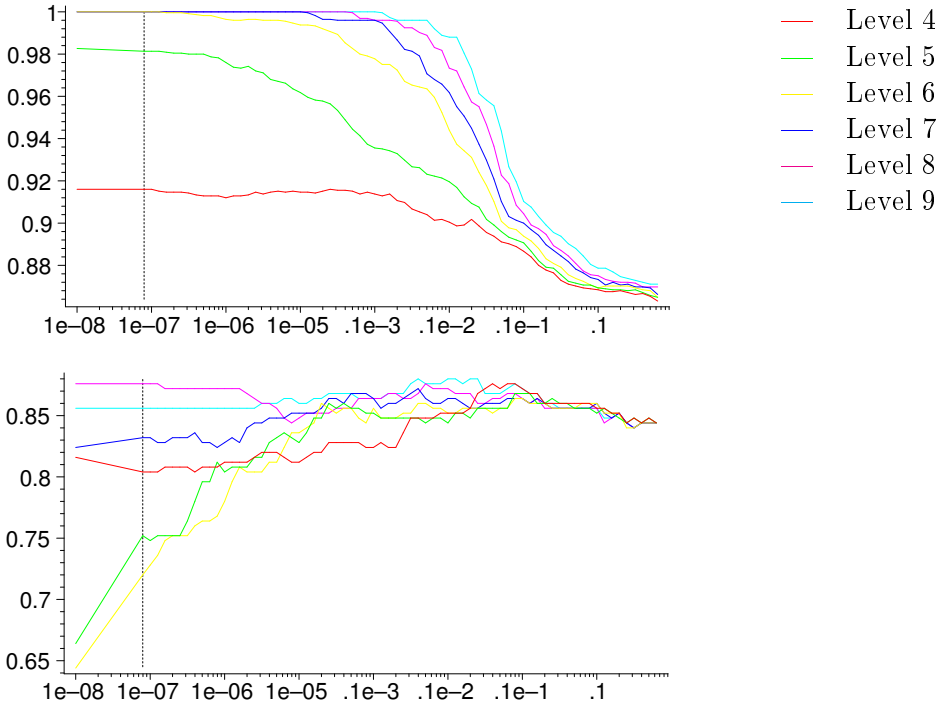


Abbildung 6.13: Trainings- (a) und Testgenauigkeit (b) in Abhängigkeit von λ (logarithmische Skala) für 10-fache Kreuzauswertung, direkte FE-Methode.

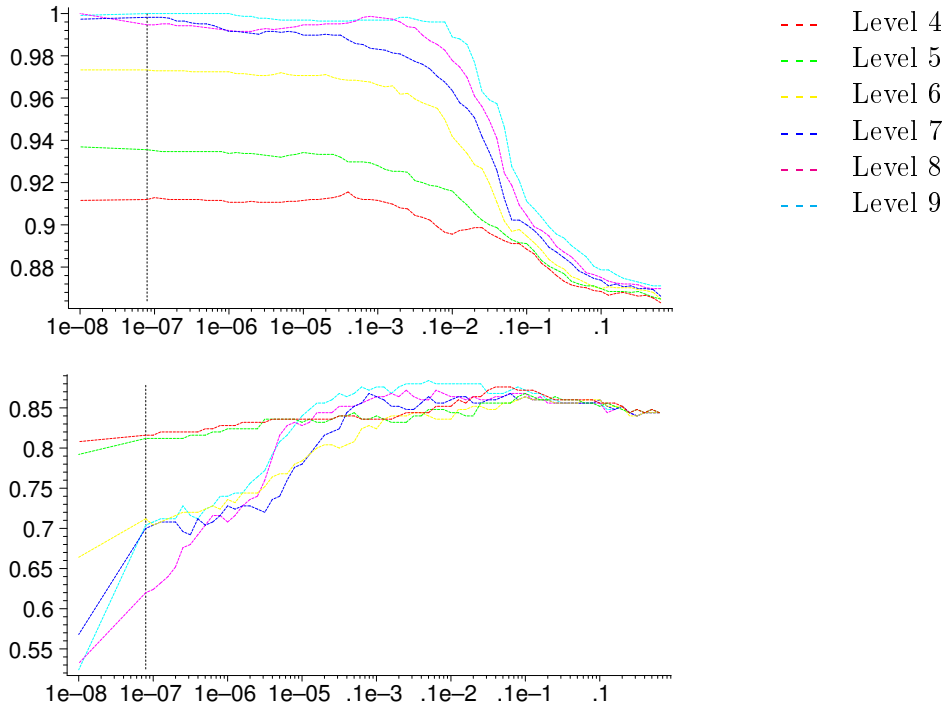


Abbildung 6.14: Trainings- (a) und Testgenauigkeit (b) in Abhängigkeit von λ (logarithmische Skala) für 10-fache Kreuzauswertung, Kombinationsmethode.

gelernt werden. Die maximal erreichbaren Werte unterscheiden sich für die beiden Techniken auf Level sechs um mehr als vier Prozent.

Im Hinblick auf die Testgenauigkeitswerte für die zehnfache Kreuzauswertung auf den Trainingsdatensatz sind für Werte des Regularisierungsparameters größer als etwa 10^{-6} kaum Unterschiede feststellbar: Für beide Techniken und für alle Level werden ähnliche Ergebnisse erzielt. Grund ist die einfache Struktur des Datensatzes.

Anders als bei dem Schachbrett- und dem Spiraldatensatz werden die besten Klassifikatoren, die durch die zehnfache Kreuzauswertung auf den Trainingsdatensatz ermittelt wurden, nun auf den Testdatensatz angewendet. Tabelle 6.3 zeigt sowohl die besten erzielten Werte der Testgenauigkeiten der Kreuzauswertung, als auch die zugehörigen Genauigkeiten für den Testdatensatz.

Level	Testgenauigkeit FE-Methode			Testgenauigkeit Komb.-Methode		
	λ	Trainings-	Testdaten	λ	Trainings-	Testdaten
4	0.003981	0.872	0.851	0.003981	0.872	0.851
5	0.012589	0.86	0.855	0.01	0.86	0.854
6	0.007943	0.852	0.854	0.01	0.852	0.854
7	0.000398	0.876	0.812	7.94328e-5	0.86	0.795
8	7.94328e-8	0.872	0.789	0.000631	0.868	0.817
9	0.002512	0.88	0.837	0.000501	0.872	0.817

Tabelle 6.3: Testgenauigkeitswerte für die direkte FE-Methode und die Kombinationsmethode. Ermittelter Wert von λ und Testgenauigkeit aus der zehnfachen Kreuzauswertung (Trainingsdaten), sowie Testgenauigkeit für die Testdaten.

Das beste Ergebnis für die Testdaten wurde mit der direkten Dünngittertechnik auf Level fünf mit 85.5% erzielt. Es genügt ein geringes Level, um die Struktur des Datensatzes zu

erfassen. Abbildung 6.15 zeigt die gelernte Funktion und die Trainingsdaten auf Level fünf mit $\lambda = 0.12589$. Der Klassifikator kann die Struktur gut erfassen.

Bei $x_1 = 0$ und $x_2 = 1$ sind die Funktionswerte der Funktion in Abbildung 6.15 sehr nahe bei null: In dem Trainingsdatensatz gibt es in diesem Bereich keine Punkte. Der betragsmäßig kleine Funktionswert des Klassifikators für Punkte in diesem Bereich spiegelt das geringe Vertrauen des Klassifikators in eine richtige Klassifikation wieder. Abbildung 6.16 (a) zeigt die zugehörige Klassifizierung des Gebietes.

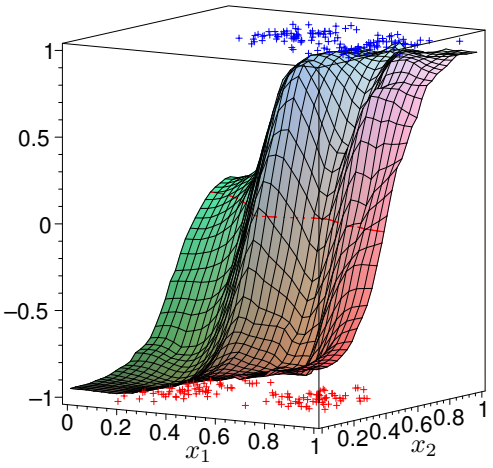
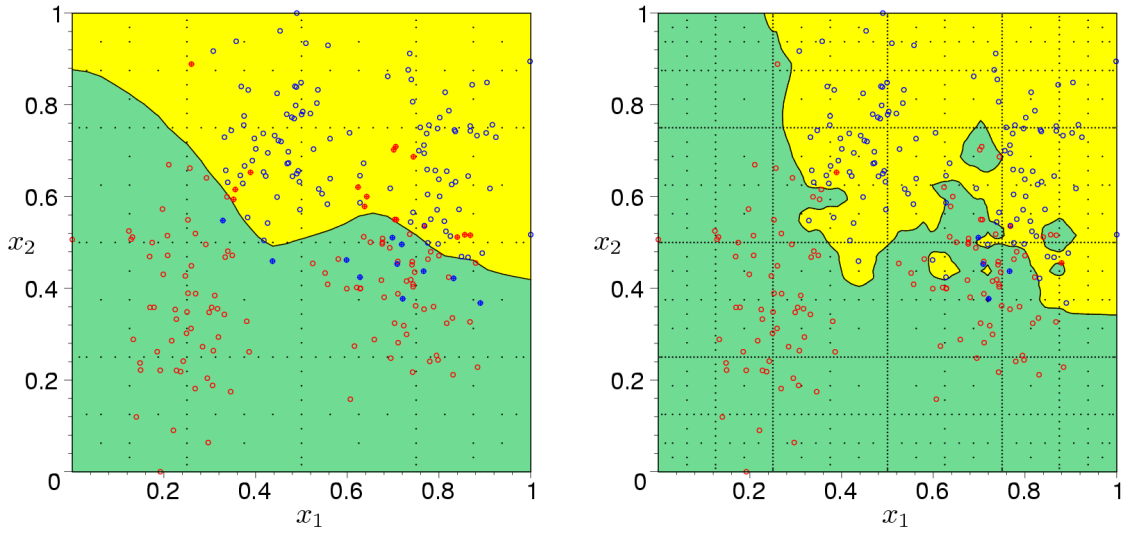


Abbildung 6.15: Klassifikator für Level 5, $\lambda = 0.12589$, direkte FE-Technik.



Level 4 Level 7

Abbildung 6.16: Klassifikationsgrenzen für Level 4 (a) und Level 7 (b), direkte FE-Methode.

Für die Level größer fünf werden im Hinblick auf den Testdatensatz zu kleine Werte von λ ermittelt. Je kleiner der Wert für den Regularisierungsparameter ist, desto schlechter ist die Genauigkeit für die Testdaten. Hinzu kommt, dass für feinere Gitter Unregelmäßigkeiten in den Trainingsdaten besser dargestellt werden können. Es tritt Überanpassung auf. Fehler im Trainingsdatensatz werden gelernt, was für den Testdatensatz zu Fehlklassifikationen führt. Abbildung 6.16 (b) zeigt die Klassifikation des gelernten Klassifikators auf Level sieben: Punkte, die durch Rauschen entstanden sind, werden berücksichtigt, und die Klassifikationsgrenzen werden zu genau aufgelöst.

7 Zusammenfassung und Ausblick

Denn unser Wissen ist Stückwerk,
und unser prophetisches Reden ist Stückwerk.
1. Korinther 13, 9

In dieser Arbeit wurde die Realisierung eines Systems in dem Computeralgebra-System Maple zur Klassifikation mit dünnen Gittern vorgestellt. Da es als Testplattform verwendet werden soll, liegt die Stärke des entstandenen Maple-Pakets eher in seiner hohen Flexibilität als in kurzen Rechenzeiten für einzelne Klassifikationsprobleme. Dennoch wurde auf eine effiziente Implementierung geachtet und es wurden für den Ansatzraum der stückweise d-linearen Funktionen geeignete Algorithmen vorgestellt.

Angewendet auf drei Testbeispiele zeigte sich, dass analog zu den Ergebnissen für die Kombinationsmethode bei [GaGT01] auch die direkte Finite-Elemente-Technik zur Klassifikation verwendet werden kann. Sie ist insbesondere in der Lage, komplexe Strukturen zu erlernen. Um Probleme mit einer einfachen Struktur zu klassifizieren, können bereits dünne Gitter geringer Tiefe genügen.

Der Aufwand wächst in der Anzahl der Trainingsdaten nur linear. Für große Datensätze ist dies ein ernstzunehmender Vorteil bezüglich manch anderen Klassifikationsalgorithmen. Der Lernalgorithmus für Support-Vektor-Maschinen skaliert beispielsweise quadratisch in der Anzahl der Trainingsdaten.

Aus der Sicht eines Anwenders ist die geringe Anzahl von Parametern ein praktischer Vorteil der Dünngitterklassifikation gegenüber anderen Verfahren. Für die Wahl des Levels kommen meist nur wenige Werte in Frage. Allein der Regularisierungsparameter bedarf besonderer Beachtung. Im Vergleich hierzu gibt es zum Beispiel bei der Verwendung Neuronaler Netze eine Vielzahl von Parametern. Beispielsweise muss im Allgemeinen die Zahl der Schichten sowie die Zahl der verwendeten Neuronen für jede Schicht abhängig vom jeweiligen Klassifikationsproblem gewählt werden.

Die Verwendung der direkten Finite-Elemente-Technik erwies sich bei der Aufgabe der Klassifikation als vorteilhaft gegenüber der Verwendung der Kombinationstechnik: Die direkte Technik zeigt zum einen eine leicht bessere Genauigkeit als die Kombinationstechnik. Besonders bei Klassifikationsaufgaben mit komplexen Strukturen führt sie zu besseren Genauigkeitswerten. Zum anderen stellte sich bei der Identifikation des Regularisierungsparameters λ heraus, dass die direkte Dünngittertechnik eine höhere Robustheit gegenüber der Wahl von λ besitzt. Das Finden des optimalen Wertes für λ ist nicht so kritisch wie bei der Kombinationstechnik. Dies erleichtert den Einsatz in der Praxis.

Zudem ermöglicht die Verwendung der direkten Dünngittertechnik zahlreiche Erweiterungen, die bei Verwendung der Kombinationstechnik nicht ohne Weiteres möglich sind. Von diesen lassen sich weitere Verbesserungen versprechen:

Beispielsweise ermöglicht die direkten Finite-Elemente-Technik den Einsatz von Funktionen höherer Ordnung, die in anderen Anwendungsgebieten bereits erfolgreich eingesetzt wurden.

Des Weiteren wird der Einsatz von weiteren Gittern ermöglicht. Denkbar sind hierfür beispielsweise bezüglich der Energienorm optimierte dünne Gitter.

Bisher werden für das Erlernen komplexer Strukturen in der Nähe des Randes des Gebietes dünne Gitter mit hohem Level benötigt. Gleiches gilt für Strukturen zwischen Gebietsmittelpunkt und den Eckpunkten des betrachteten Gebietes. Vielversprechend ist hier die Verwendung von adaptiven dünnen Gittern. Über eine problemangepasste Verteilung der Gitterpunkte könnte die Zahl der benötigten Freiheitsgrade deutlich reduziert werden.

Das in dieser Arbeit vorgestellte System zur Klassifikation mit dünnen Gittern stellt eine Basis zur Verfügung, um diese und weitere Erweiterungen zu erproben und zu evaluieren.

Literaturverzeichnis

[Alle74] D. Allen. The relationship between variable selection and data augmentation and a method for prediction. *Technmetrics* Volume 16, 1974, p. 125–127.

[Aron50] N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.* Volume 68, 1950, p. 337–404.

[BCPP+97] I. S. Bhandari, E. Colet, J. Parker, Z. Pines, R. Pratap and K. Ramanujam. Advanced Scout: Data Mining and Knowledge Discovery in NBA Data. *Data Mining and Knowledge Discovery* 1(1), 1997, p. 121–125.

[BuGr04] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica* Volume 13, 2004, p. 147–269.

[EvPP00] T. Evgeniou, M. Pontil and T. Poggio. Regularization Networks and Support Vector Machines, 2000.

[FPSSU96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Hrsg.). *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press. 1996.

[FrPSM92] W. J. Frawley, G. Piatetsky-Shapiro and C. J. Matheus. Knowledge discovery in databases - an overview. *Ai Magazine* Volume 13, 1992, p. 57–70.

[GaGT01] J. Garcke, M. Griebel and M. Thess. Data Mining with Sparse Grids. *Computing* 67(3), 2001, p. 225–253.

[GoHW79] G. H. Golub, M. Heath and G. Wahba. Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter. Volume 21, 1979, p. 215–224.

[GrSZ92] M. Griebel, M. Schneider and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens (Hrsg.), *Iterative Methods in Linear Algebra.* IMACS, Elsevier, North Holland, 1992, p. 263–281. also as SFB Bericht, 342/19/90 A, Institut für Informatik, TU München, 1990.

[Hada23] J. Hadamard. *Lectures on Cauchy's problem in linear partial differential equations.* Yale University Press, New Haven, CT, U.S.A. 1923.

[HaMS01] D. J. Hand, H. Mannila and P. Smyth. *Principles of Data Mining.* MIT Press. 2001.

[Koha95] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *IJCAI*, 1995, p. 1137–1145.

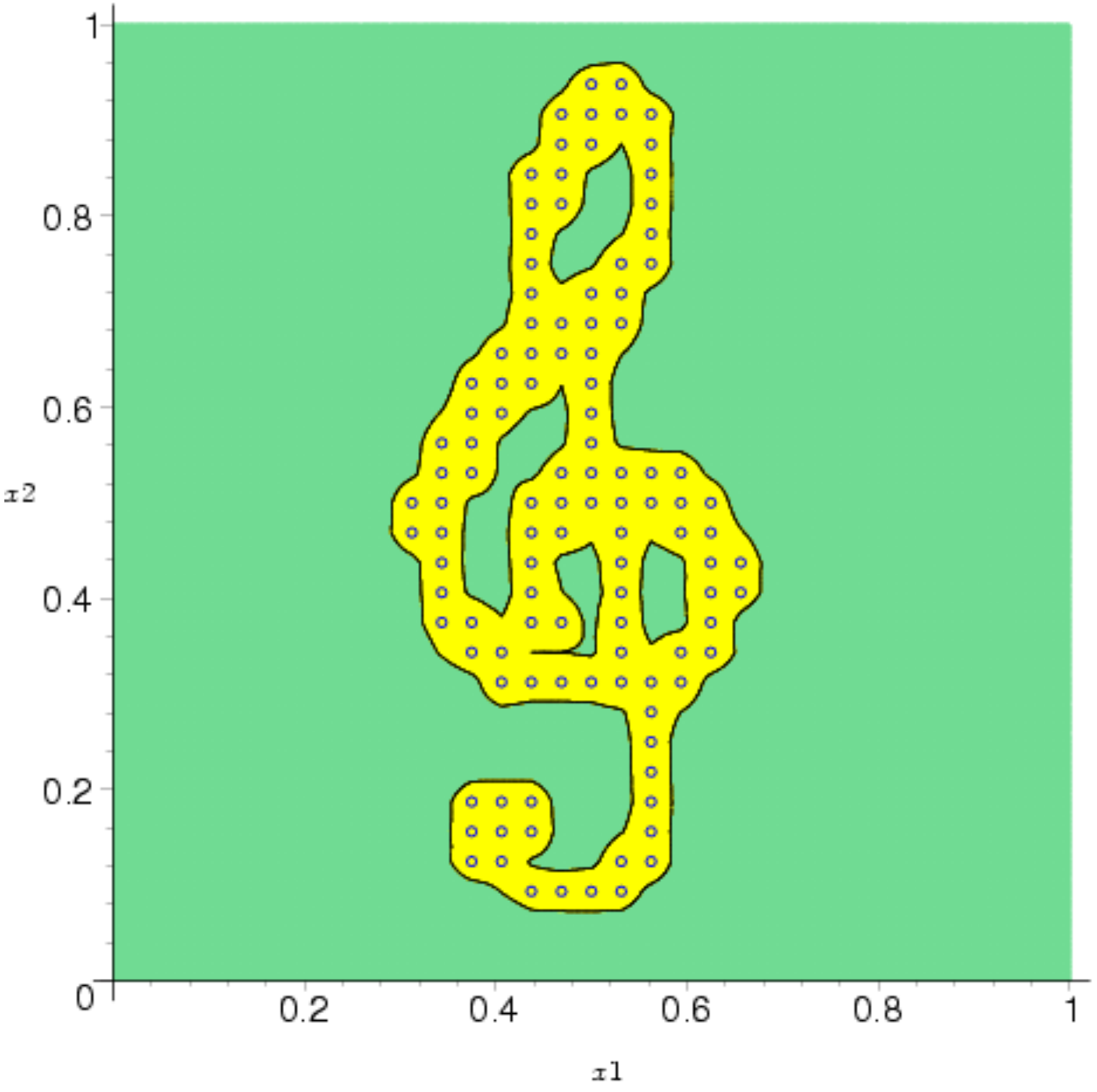
[RiHj95] B. D. Ripley and N. L. Hjort. *Pattern Recognition and Neural Networks.* Cambridge University Press, New York, NY, USA. 1995.

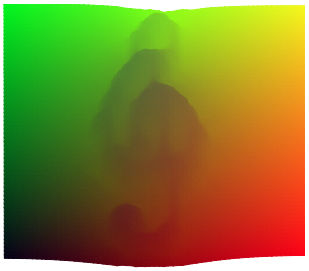
[Sing98] S. Singh. 2D spiral pattern recognition with possibilistic measures. *Pattern Recogn. Lett.* 19(2), 1998, p. 141–147.

[TiAr77] A. Tikhonov and V. Arsenin. *Solutions of ill-posed problems.* W.H. Winston, Washington, D.C. 1977.

[Vapn82] V. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, Berlin. 1982.

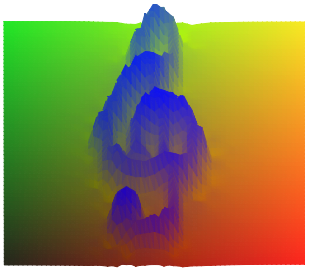
[Vapn95] V. N. Vapnik. *The nature of statistical learning theory.* Springer-Verlag New York, Inc. 1995.


1
0.8
0.6
x2
0.4
0.2
0
0.2
0.4
0.6
0.8
1
x1

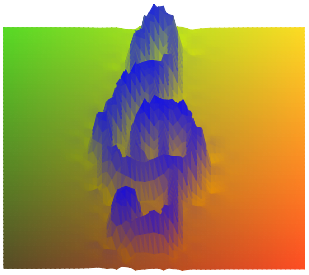


$\lambda = 0.1$

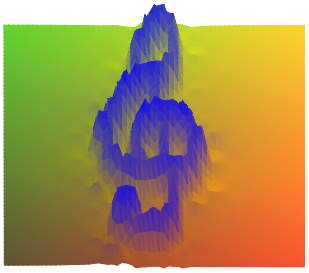
$\lambda = 0.01$



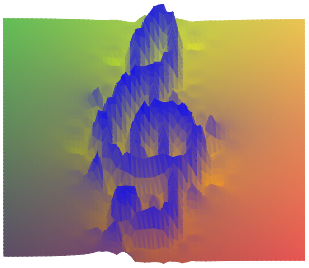
$\lambda = 0.005$



$\lambda = 0.001$

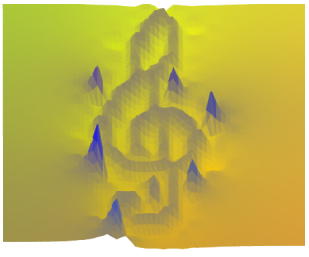


$\lambda = 0.0005$



$\lambda = 0.0001$

$\lambda = 0.00001$



$\lambda = 10^{-15}$

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfasst und nur die
angegebenen Quellen benutzt zu haben.

_____

(Dirk Pflüger)